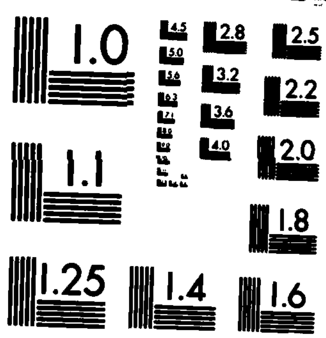
AD-A153 249 A MODERN CONTROL THEORY ENHANCEMENT TO AN INTERACTIVE CONTROL ENGINEERING..(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. C NARATHONG 1/2

UNCLASSIFIED DEC 84 AFIT/GE/ENG/84D-49 F/G 9/2 NL



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

DTIC

1

AD-A153 249

AF INSTITUTE OF TECHNOLOGY

DTIC FILE COPY

DISTRIBUTION STATEMENT A

Approved for public release;
Distribution Unlimited

DEPARTMENT OF THE AIR FORCE

AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

85 4 05 032

[illegible]

[illegible]

AFIT/GE/ENG/84D-49

A MODERN CONTROL THEORY ENHANCEMENT TO AN INTERACTIVE CONTROL ENGINEERING COMPUTER ANALYSIS PACKAGE (ICECAP)

THESIS

Presented to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Air Training Command

in Partial Fulfillment of the

Requirements for the Degree of

Master of Science

by

Chiewcharn Narathong, B.S.

2d Lt. RTAF

Graduate Electrical Engineering

December 1984

PREFACE

This thesis effort continues the development of the Interactive Control Engineering Computer Analysis Package (ICECAP). This investigation expanded functional capabilities of ICECAP involved in modern control theory. As a result, ICECAP is now able to perform both modern control functions and matrix operations in addition to continuous time control functions.

I wish to express my sincerest gratitude to my thesis advisors, Dr. John Jones and Dr. Robert E. Fontana. Their encouragement, suggestions, and ready assistance have proved to be the key successfulness of this thesis. Especially, Dr. Fontana who spent many hours in reviewing and correcting my writing deserves my deepest appreciation. I also wish to thank the other members of my thesis committee, Dr. Constantine H. Houpis and Dr. Peter S. Maybeck, for their efforts in reviewing and critiquing my thesis. Finally, I would like to express my sincerely appreciation to my fiancee, Kai, for her encouragement and constant love thoughout my entire AFIT assignment.

CONTENTS

CONTENTS

CONTENTS

CONTENTS

LIST OF FIGURES

LIST OF FIGURES

LIST OF TABLES

AFIT/GE/ENG/84D-49

ABSTRACT

This thesis reports on a continued development of an Interactive Control Engineering Computer Analysis Package (ICECAP). This package applies to discrete and continuous systems. The effort developed and [illegible] functional requirements involved in modern control theory. This includes implementing [illegible] [illegible] the steady [illegible] Riccati solver, [illegible] conditions, [illegible] G(s) [illegible] function, and [illegible] function.

Major [illegible] Riccati solver. [illegible]

Riccati [illegible] VAX 11/780 [illegible] [illegible] implementation [illegible]

CHAPTER 1

INTRODUCTION

1. Background

Control systems design and synthesis are very complicated engineering processes. There are many steps involved in repetitive mathematical manipulations such as those found in calculus, matrix algebra, transformations between frequency and time domains, graphical plotting, and probability and stochastic process characterization and simulation. Conceptually, most of these computations are very simple. However, they require a significant amount of time and effort to perform. Thus, such calculations can be tedious if accomplished manually. To allow control engineers to concentrate on the critical aspects of their design without spending a great deal of time on tedious calculations and graphical techniques, tools such as computer-aided design (CAD) programs are necessary.

Over the years, industries and universities have developed several CAD programs to assist control engineers and students. The Air Force Institute of Technology (AFIT), like other universities, has developed CAD research programs for control system design and analysis. One of the most successful products of the project is TOTAL [2]. TOTAL is an interactive software package for digital and continuous control system analysis and synthesis. It was

initially developed by two AFIT students, Frederick O'Brien [1] and Stanley Larimer [2]. Both students consolidated several AFIT computer programs into TOTAL and hosted it on the Control Data Corporation (CDC) Cyber computer at Wright-Patterson Air Force Base (WPAFB), Ohio. In a later effort, Glen Logan [3] successfully transported TOTAL from the Cyber to a VAX 11/780, an AFIT/EN computer located in the Information Sciences Laboratory. His effort resulted in improving the user interfaces for TOTAL and providing a backup CAD capability, thus minimizing the impact of system down time. Logan's work is known as VAXTOTAL. Other efforts were recently accomplished by two students, Charles Gembarowski [4] and Robert Wilson [5]. Both have improved VAXTOTAL's interactive environment by incorporating many of its routines into a highly structured and modular program known today as ICECAP (Interactive Control Engineering Computer Analysis Package), currently hosted on the VAX 11/780. Their efforts resulted in a new system which provides both user "friendliness" and the ongoing maintenance of the system. ICECAP is now able to perform an entire range of continuous time control functions.

2. Problem

Although TOTAL performs very well as far as problem solving is concerned, it still has several deficiencies that reduce the overall effectiveness of the program. These

deficiencies fall into four major areas :

2.1 Structure. TOTAL has a very tight structure which makes the program difficult to expand. The reason for this is that TOTAL is composed of many existing FORTRAN routines [5]. Each one of them uses FORTRAN COMMON variables which make the program very tightly coupled [4]. Additionally, some external routines such as the CALCOMP and PLOT10 libraries are not identified [4]. This makes the transfer of the program to another computer system difficult. Finally, TOTAL is not well-documented in source code, thus causing difficulty in improving and maintaining the system.

2.2 Usability. TOTAL is not well-designed with respect to user interfaces. It has over 100 options in which the user interactively enters and manipulates transfer functions and matrices, while appling root-locus, frequency and time response options. This makes TOTAL inconvenient to use, especially for the inexperienced user.

2.3 Workload of Computer. Since the CDC Cyber is one of the most heavily used computer facilities at WPAFB, it is often saturated. When this happens, it may be impossible to access the system. A backup capability using the VAX for courses and design applications will serve to reduce the Cyber workload.

2.4 Computational Problem. Lastly, and more importantly, the "number crunching" foundations of TOTAL are

poor from a numerical precision point of view. Numerical precision, numerical stability and computational efficiency need improvement. Complete rework of this core is required for high dimension.

To overcome these problems, ICECAP was developed with certain stages. First, it was designed in a highly structured and modular fashion so that subsequent improvement and maintenance can be easily accomplished. Additionally, it was also designed to be a system which is easier to learn and use for both experienced and inexperienced personnel. Finally, as a released product, ICECAP will be rehosted on the AFIT/EN Scientific Support Computer Facility, a VAX which operates under the VMS operating system. This system can accommodate a large student load and is intended for student class use. Once ICECAP is rehosted, students and faculty will be able to use it in addition to Cyber [illegible].

3. Scope

This thesis investigation addresses the continued development and implementation of ICECAP functional requirements involved in modern control theory. This includes matrix operations and solving optimal control problems which involve in solving a continuous time steady state Riccati equation. Additionally, a $C(sI - A)^{-1}B$ function, polynomial operations, and a state variable

feedback design using phase variable representation will be developed and implemented. Notice that some of these functions are already available in TOTAL. However, this thesis effort will develop routines that perform more efficiently.

4. Summary of Current Knowledge

Currently, there are existing computer programs [2,7,8,15,35,36,39,41] that accomplish matrix operations and the basics of modern control design and analysis. Unfortunately, few of these programs provide the user with an integrated package of design tools to perform the wide range of computations required for extensive work in modern control system design. In addition, most of these programs are hosted on different computer systems.

5. Standards

The developed routines must provide results consistent with testing requirements which shall be described in Chapter 2. Additionally, the routines must be well-documented for subsequent developments. Finally, they shall provide a well-designed human interface with so-call "user friendliness" [6].

6. Approach

The following steps were followed during this thesis investigation.

1. Literature Research. Study and review of the

literature on CAD and synthesis in modern control areas. This included study of tools investigated by previous theses [1,2,3,4,5] as well as tools found during this thesis investigation [7,8,15,35,36,39,40,41]. Additionally, the current version of ICECAP and TOTAL's routines will be closely studied in detail. Finally, the use of VAX/VMS operating system must be understood.

2. Requirements Analysis, and Routine Selection. First, determine which functions are necessary in the system and which options are of primary importance and require early implementation. Second, analyze the existing routines studied in step 1 and select the most suitable ones. This is based mainly upon efficiency, numerical precision, numerical stability, and adequate performance of routines.

3. Development and Implementation. This phase consists of the following steps :

3.1 Modify the routines selected in step 2 to meet ICECAP's structure.

3.2 Develop the additional routines to meet the modern control functional requirements as described Chapter 2.

3.3 Incorporate the modified and developed routines into ICECAP.

3.4 Test and modify if necessary in order to meet the testing requirements which are described in Chapter 2.

4. Design Documentation. Document all routines modified and developed during this investigation. This

documentation will be included in Appendix B. This will serve as a convenient and useful documentation for subsequent development of ICECAP's project.

7. Material and Equipment

All materials and equipments required to perform this study are currently available in the Information Sciences Laboratory of the AFIT, School of Engineering.

8. Overview of Thesis.

Instead of including all source code lists of modules developed during this investigation, this thesis consists of major chapters and appendices which contain documentation of the developed modules. This shall be useful for follow-on efforts. (Note : A complete source code listing of all development modules is maintained in the AFIT Information Sciences Laboratory. It can be generated whenever needed by utilizing the information contained in Appendix B.)

In the interest of continuity, the first two chapters of this thesis are structurally similar to Wilson's thesis [5]. A brief summary of each chapter and appendix is as follows :

Chapter 2 summarizes the requirements definition that were first described in Reference [5]. Major emphasis is placed on the modern control functional requirements, since this thesis develops and implements subroutines to meet these requirements. The definition and the determination of

priority for the modern control functional requirements portion are also described in this chapter. Finally, the testing requirements for the entire system are presented.

Chapter 3 reviews ICECAP's structure and investigates the various routines and algorithms available for this project. The specific selections for these items are established in this chapter and the reasons for those selections are discussed and analyzed.

Chapter 4 discusses the system design in a broad overview fashion. The selected routines and algorithms are also presented explicitly. Finally, it describes the process of incorporating the developed routines into ICECAP.

Chapter 5 details the ICECAP testing phase of the investigation and presents the test results. The testing philosophy is discussed followed by the testing of the developed functions.

Chapter 6 presents the conclusions and recommendations of this thesis effort.

Appendix A contains the railroad diagrams. This railroad diagrams explain a formal description of the acceptable language used in performing matrix operations.

Appendix B contains the subroutine descriptions of non-trivial routines developed during this study.

Appendix C contains the ICECAP user's manual which includes the overview of ICECAP, ICECAP commands, and example problems.

CHAPTER 2

REQUIREMENTS

1. Introduction

As mentioned in the Chapter 1, the main objective of this study is to continue development and implementation of ICECAP functional requirements involved in modern control theory. For the sake of continuity, this chapter first repeats the definitions of priorities and the annotation standards of requirements as defined in Chapter 2 of [5]. Once these are established, the functional requirements themselves are presented. These include those listed in Chapter 2 of [5] and those established during this investigation. Additionally, the human engineering and software engineering requirements described in Chapter 2 of [5] are also repeated here. This serves as a quick reference of ICECAP's current state for follow-on efforts. After providing a historical perspective, this chapter addresses the priorities that are structured so that ICECAP is capable of solving fundamental modern control problems. Finally, the testing requirements for the entire system are discussed.

2. Definition of Priorities

The meanings of requirements priorities have been categorized into one or more of three priorities in order to facilitate partitioning the entire ICECAP project into a meaningful development. The definitions of these priorities

are described as follows :

2.1 Priority One. This priority means that the requirement must be at least partially contained in the initial program design in order to have a running program with which to demonstrate both feasibility and capability. This may involve both requirements that are already satisfied by VAXTOTAL [3] and requirements that are new to ICECAP. This category generally includes all of the human interface requirements, the incorporation of "on-line" help/teach modules for the most important features required for continuous time design and analysis, and software engineering requirements. Continuous time features are considered a higher priority than the discrete time features because students are normally taught the fundamentals of continuous control systems prior to the study of discrete control systems.

2.2 Priority Two. This priority includes requirements that are at least partially implemented in VAXTOTAL [3] but need not be initially implemented in ICECAP in order to demonstate feasibility and capability. This category generally refers to matrix manipulation features, and discrete time design and analysis features.

2.3 Priority Three. This category includes functional requirements needed to have a complete control system computer-aided design package. It includes functional

requirements related to the control system design area presently within the state of the art but not yet implemented. Examples are the stochastic estimation and control requirements.

3. Requirements.

Definition. ICECAP will assist the user in performing conventional, modern, and stochastic control system design and analysis for both discrete and continuous systems.

Annotation Standards. ICECAP's requirements are annotated in the following format : (priority classification(s), status) [reference(s)]. The priority classification(s) is a number in the range 1-3. The status is reflected in a capital letter if the requirement has been fully implemented on ICECAP's structure; otherwise, the letter appears in lowercase. The letters used are A, for Gembarowski [4]; B, for Wilson [5]; C, for Narathong [this thesis]; D, for Armold [46]; (E will be for the next follow-on effort, then F etc.). Requirements yet to be addressed are assigned the status "TBD" (To Be Determined). The references refer to corresponding entries in the bibliography. The interested reader can find the theoretical principles that "spawned" the system requirements by reading the cited references.

Listing. ICECAP's requirements can be grouped into three main categories -- (1) functional, (2) human

engineering, (3) software engineering. The specific requirements (along with their associate priorities, implementation status, and applicable references) are as follows :

3.1 Functional Requirements. ICECAP shall provide the following functional capabilities :

3.1.1 Conventional Control

- Open Loop Transfer Function (1,A) [34,37]
- Closed Loop Transfer Function (1,A) [34,37]
- Forward Transfer Function (1,A) [34,37]
- Feedback Transfer Function (1,A) [34,37]
- Steady State Response Analysis (1,A,B) [34,37]
- Transient Response Analysis (1,A,B) [34,37]
- Partial Fraction Expansion (1,B) [34,37]
- Root Locus Analysis (1,A,B) [34,37]
- Laplace Transformation (2,TBD) [34,37,38]
- Inverse Laplace Transformation (2,B) [34,37,38]
- Frequency Response Evaluation (2,B) [34,37]
- Time Response Plot (2,B) [34]
- Bode Plot (2,B) [34,37]
- Direct Polar Plot (2,TBD) [34,37]
- Inverse Polar Plot (2,TBD) [34,37]
- Nyquist Criteria (2,TBD) [34,37]
- Nichols Plots (2,TBD) [34,37]

- Guillemin-Truxal Design (2,a,b) [34]
- Block Diagram Manipulation (2,A,B) [34,37]

3.1.2 Modern Control

Matrix Analysis [9,10,11,18,20,24,25,26,27,28,32]

- Matrix Arithmetic
  - -- Matrix right division (2,C)
  - -- Matrix left division (2,C)
  - -- Matrix transpose (2,C)
  - -- Matrix addition (2,C)
  - -- Matrix subtraction (2,C)
  - -- Matrix multiplication (2,C)
  - -- Raises matrix to powers (2,C)
- Basic Properties
  - -- Condition number in 2-norm (2,C)
  - -- Determinant (2,C)
  - -- Norm (2,C)
  - -- Rank (2,C)
  - -- Inverse (2,C)
  - -- Pseudoinverse with optional tolerance (2,C)
  - -- Inverse Hilbert matrices (2,C)
  - -- Kronecker tensor product (2,C)
- Trancendental Matrix Functions
  - -- Arctangent (2,C)
  - -- Cosine (2,C)
  - -- Matrix Exponential (2,C)

-- Natural logarithm (2,C)

-- Sine (2,C)

-- Square root (2,C)

- Decomposition and Factorizations

-- Cholesky factorization (2,C)

-- Eigenvalues and Eigenvectors (2,C)

-- Hessenberg or Tridiagonal Form (2,C)

-- Factor from Gaussian Elimination (2,C)

-- Orthogonal vectors spanning range (2,C)

-- Schur triangular form (2,C)

-- Reduced row echelon form (2,C)

-- Singular value decomposition (2,C)

Polynomial Operations

- Addition (2,C)

- Subtraction (2,C)

- Multiplication (2,C)

- Division (2,c)

- Raise polynomial to powers (2,C)

Optimal Control Design [12,22,23,37]

- Algebraic Riccati Equation (ARE) solver (2,C)

- Riccati Differential Equation Solver (2,c) [12]

Transfer Function via $C(SI - A)^{-1}B$ (2,C)

State Feedback Design [34]

- Phase Variable Representation (2,C)

REQUIREMENTS

- Physical Variable Representation (2,c)
- Observable Variable Representation (2,c)
- Generalized Control Canonical Form (2,c)

Linear Regulator, Observers, and Trackers [34]

- Using Entire Eigen Structure Assignment (2,c)
- Using LQG Design (optimal deterministic control and Kalman filters) (3,TBD) [31]

Controller Robustness (2,TBD) [31]

Simulation (Performance Against a "Truth Model")

- Singular Value versus Frequency Plots (3, TBD) [31]
- Structural Singular Values (3,TBD) [31]

3.1.3 Discrete Conventional Control

Z-Transformation (2,D) [33]

Inverse Z-Transformation (2, D) [33]

Digital Computer [illegible] (2,[illegible]) [33]

Z-[illegible] (2,[illegible]) [33]

S-[illegible] (2,[illegible]) [33]

S-[illegible]

- [illegible]
- [illegible]

[illegible]

Plots (Z, W, S primary strip, time) (2,TBD) [33]

Simulation (Performance Against a "Truth Model") (3,TBD) [29,31]

3.1.4 Stochastic Estimation and Control Design

Kalman Filter Design for Continuous and

Discrete Time Measurements (3,TBD) [29]

Analysis of Kalman Filter Design (3,TBD) [29]

Square Root Filtering (3,TBD) [29]

U-D Covariance Factorization Filtering (3,TBD) [29]

Weiner Filtering (3,TBD) [29]

Optimal Smoothing (3,TBD) [30]

LQG Controller Design for Continuous and Discrete Systems (3,TBD) [30,31]

LQG/LTR Method (3,TBD) [31]

Observer and Full-State Feedback Controller via Pole-Placement Methods (3,TBD) [30,31]

Extended Kalman Filter Design (3,TBD) [30]

Simulation (Performance Against a "Truth Model") (3,TBD) [30,31]

3.2 Human Engineering Requirements. ICECAP shall be user "friendly" in that human factors will drive the design of the human/computer interface [6].

3.2.1 ICECAP shall provide on-line assistance upon demand. (1,2,3,B,C)

3.2.2 ICECAP shall be command-oriented. It will assist the user in formulating commands. This assistance

shall not be distractive. It shall not impede those users who do not need on-line assistance. (1,B,C)

3.2.3 ICECAP shall provide instruction in the various aspects of control theory through some sort of teaching facility. (2,3,b)

3.2.4 ICECAP shall notify the users when they have erred in providing input by using meaningful error messages. (1,B)

3.2.5 ICECAP shall provide a facility for providing meaningful and selective printed output as the means of documenting the users' design of control systems. (1,B,C)

3.2.6 ICECAP shall provide a means of storing the essentials of a design in progress so that the users may continue their designs at future sessions. (1,B,C)

3.2.7 ICECAP shall provide a capability for the users to define command strings so that they may iterate a design without having to type in the same commands repeatedly. This shall include a facility for the users to specify data as part of the command string. (2,TBD)

3.3 Software Engineering Requirements. ICECAP shall be designed and documented using sound software engineering principles such as those advocated in the software engineering literature [6,17] so that the program can be easily maintained and augmented. The following requirements

address these concerns :

3.3.1 ICECAP shall be as portable as reasonable (i.e. capable of being rehosted on other VAX's). (1,C)

3.3.2 ICECAP shall be modular. (1,A,B,C)

3.3.3 ICECAP shall use loosely coupled modules as much as possible. (1,a,b,c)

4. Priority Definition and Determination

Due to time constraint, this one project cannot develop the modules that solve all aspects of modern control problems. Thus, it is necessary to facilitate partitioning the modern control functional requirements [described in Section 3.1.2] into priorities. The priorities are structured so that ICECAP is capable of solving basic modern control problems. Such functions are included in the framework of priority one modules. Priority two then follows and consists of the remainder of the modern control modules. The following list indicates the initial priority established for ICECAP's modern control functional requirements.

Priority One Modules. These are the priority one requirements which partially come from Section 3.1.2. These modules are essential for basic modern control system work.

- All requirements for matrix analysis
- Polynomial Operations
- Transfer function via $C(SI - A)^{-1}B$

REQUIREMENTS

- Algebraic Riccati Equation (ARE) Solver
- State Variable Feedback Design
  - -- Using Phase Variable Representation

Generally, matrix and polynomial operations are considered part of modern control theory and are presented here as priority one. The reason for this is that they are basic mathematical tools useful in many disciplines. A system that provides assistance with this process would be quite beneficial and warrants assignment of a higher priority than that associated with modern control. Transfer function via $C(SI - A)^{-1}B$, ARE solver, and state variable feedback design using phase variable representation are generally considered to be a fundamental modern control design concept and are usually taught before other advanced modern control designs. They are therefore included in priority one.

Priority Two Modules. This priority consists of the remainder of the modern control functional requirements described in Section 3.1.2. These modules enhance advanced modern control design and provide additional design and synthesis capability.

These requirements may not be complete. However, they are representative of the kinds of essential functions for the system to be useful as a modern control design tool.

5. Testing Requirements.

This section discusses the testing requirements of the ICECAP that were first described in Chapter 2 of Reference [5]. Modifications are made to the modern control functional testing requirements, in order to ensure that the modules developed during this study provide validation of numerical results.

5.1 Functional Requirements. The modern control functional requirements that have been implemented shall be tested using known test cases, such as those examples presented in References [8,12,34,39]. The numerical results are expected to have small discrepancies due to the numerical round-off error in computer wordlength. Note that ICECAP is hosted on a VAX 11/780 which has only a 32-bit wordlength.

5.2 Program Flow. All developed programs shall be tested to determine whether or not the program runs properly. The program must transition to valid known states. The program must not hang in an endless loop. The ability to exit "gracefully" from the program must be demonstated.

5.3 On-line Assistance. The ability of the intended user to formulate the commands necessary to design and analyze a control system with on-line assistance that is provided must be demonstrated.

5.4 Output Capability. The ability to document and store a control system design and analysis session selectively and conveniently must be demonstrated.

6. Summary

The system requirements specification for the ICECAP project is presented in accordance with Reference [5]. Priority categories and the status of implementation classifications were documented by Wilson's effort [5]. This thesis adds the modern control functional requirements that were not initially included in previous development. Additionally, the definition and determination of priority for the modern control requirements portion are presented explicitly. Finally, the testing requirements initially defined by Wilson [5] are presented.

CHAPTER 3

ICECAP's STRUCTRUE, ROUTINES AND ALGORITHM SELECTION

1. Introduction

In designing ICECAP, emphasis was placed on making the system "user-friendly". For example, many of the prompting features were integrated into the system so that a new user can quickly learn and really "enjoy" using it.

In contrast to the past efforts [3,4,5], greater emphasis is placed on expanding the functional capabilities of the system. In particular, the modern control functional capability is developed and implemented. This chapter begins with a review of the previous design as it relates to the Human Interface and its component areas. It then discusses the routine and algorithm selection which are somewhat dictated by previous designs and other contraints such as cost and procurement of the routines.

2. ICECAP's Structure

The design foundation for ICECAP relies on the following requirements as stated in Chapter 2.

-- The system shall be user-friendly.

-- The system shall be easy to use.

This section reviews the design of the system as its relates to the Human Interface and its component areas. These areas have direct impact on incorporating the new modules on the system. Other areas of the design were

discussed in [5]. The specific areas reviewed are:

2.1 Program Control - how the user manipulates execution of the program.

2.2 Help - on line assistance

2.3 Information Transfer - the program's response to the user

2.4 Program Parameter Control - the user's ability to view and alter the system control parameter

2.1 Program Control

ICECAP was designed to have the user entering commands into the system which then cause the system to execute its function. Each word was selected so as to define the intended action accurately. Each action has one or several words associated with it. For instance, the command words necessary for plotting a root locus are "PLOT ROOT LOCUS". The system decodes the words and, once decoded, the commands are translated. The translation is then passed to a set of modules that actually execute the command. The overall program transform was developed by Logan [3] as shown in Figure 1.

2.2 Help Capability

ICECAP was designed to be user friendly. An aspect which helps ICECAP become user friendly is the availability of on-line assistance. Help is provided in the sense of "help upon demand". The design structure for the help process was discussed in [5] and is repeated here for continuity.

2.2.1 Invalid Command Response. If the user has entered an invalid command, the system responds with a

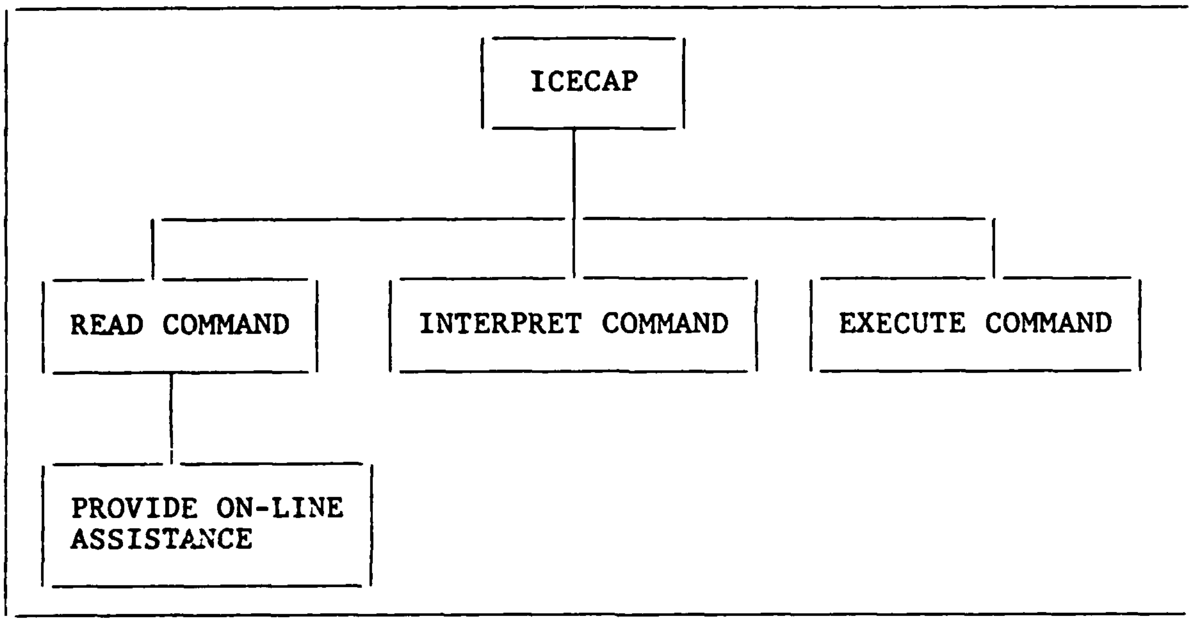


FIGURE 1. ICECAP's FRONT-END STRUCTURE

message as to the nature of the error and then the user is given the opportunity to reenter the erroneous portion.

2.2.2 Incomplete Command Response. If the user has entered an incomplete command, the system responds with choices for the next command word. The design structure allows a short explanation of the nature of each of the choices. The user then simply types in one of the listed choices.

2.2.3 Complete and Valid Command. If the user has

entered a command that is both complete and valid, the system responds by executing the command.

2.3 Information Transfer

The information that ICECAP transfers to the user consists of data such as points on a root locus diagram, frequency and/or phase response plot, and figures of merit. These data are conveyed to the user by using any of the standard output available to the computer such as the CRT and line printer. ICECAP also offers a choice of presenting the information in tabular or graphical form. Additionally, it allows the user to review all outputs before printing out on the line printer. Finally, ICECAP has the ability to store files of intermediate results so that the user can continue his/her work at other sessions.

2.4 Program Parameter Control

ICECAP includes system parameters which control the execution of the routines within the system as, for example, boundaries for root locus and frequency response plots, step sizes for calculating root locus, and output and input selections. Some of these parameters were hard coded into the software, but ICECAP still allows the user to change these parameters as desired. The change will affect the computations from that point on.

3. Routine and Algorithm Selection

This section discusses the selection of the new

routines for matrix analysis and algorithm for other modern control functions.

3.1 Routine Justification and Selection

There are existing FORTRAN subroutines that do matrix operations, such as, the EISPACK [36] and LINPACK [35] software packages. These two packages represent the state of the art in matrix computational methods. EISPACK is a package of over 70 FORTRAN subroutines for various matrix eigenvalue computations. LINPACK is a package of 40 FORTRAN subroutines for solving and analyzing simultaneous linear equations and related matrix problems. The subroutines from these two packages provide assistance with a basic mathematical tool that can be used to perform modern control functions. Presently, these two packages are already integrated in several computer-aided design programs, for example, CTRL-C [41], MATRIX_x [40], LQGLIB [39], and MATLAB [7]. Unfortunately, all of these programs except MATLAB are commercial and the source codes are not released. MATLAB is available without constraints. The program was originally developed by Cleve Moler at the Unversity of New Mexico. It was written as a convenient tool for computations involving matrices. Additionally, it also provides access to the LINPACK and EISPACK software. The capabilities range from standard tasks such as solving simultaneous linear equations and inverting matrices, through symmetric and nonsymmetric

eigenvalue problems, to fairly sophisticated matrix tools such as the singular value decomposition.

Based upon a comparison of cost and capabilities of existing programs, MATLAB (with the LINPACK and EISPACK subroutines) is considered the best choice. Its computation routines will be used for lower-level modules while its front-end (see Section 2 in Chapter 4 for more detail) can be used to accept input matrices.

3.2 Algorithm Selection

Again, there are existing routines within TOTAL and ICECAP that solve the Riccati equation, compute $C(SI - A)^{-1}B$, and do polynomial operations. However, these routines are somewhat out of date. For example, those routines that solve Riccati equation in OPTCON [8] were coded based upon the iterative algorithm [22]. Presently, several methods are developed based upon matrix theory. These methods can handle a large class of problem very well [12]. This thesis surveys and discusses various algorithms as follows :

3.2.1 Riccati Solver. The Riccati equation plays fundamental roles in the analysis, synthesis, and design of linear-quadratic-Gaussian control and estimation systems as well as in many other branches of applied mathematics. Due to the time limitations, this thesis only studied two of the best algorithms, namely the Schur vector approach [12] and the eigen number approach [42]. These two methods are quite

stable numerically and perform reliably on systems with dense matrices up to order 100. Unlike other methods using eigenvectors such as Potter's approach [43] and its extensions, the Schur vector approach is very fast since the reduction to RSF (Real Schur Form) is an intermediate step in computing eigenvectors (for further detail in the Schur vector approach see [12]). Another advantage of this method is its general applicability for engineering applications. The Schur vector method provides a unique solution to the solution with a very fast speed (see Table 1 in Chapter 5).

3.2.3 Polynomial Operations. Similar to the case of $C(SI - A)^{-1}B$ computations, a new algorithm for polynomial multiplication is developed. The routines in TOTAL are not used since they do not match the MATLAB front-end, and more importantly, those routines are poorly documented.

3.2.4 State Variable Feedback Design. This thesis effort develops a new algorithm to perform this function. This algorithm provides an adequate numerical accuracy (see Section 4.4 in Chapter 5 for numerical results). Chapter 4 presents this algorithm in detail.

4. Summary

This chapter discusses the previous system design and the selection of new routines and algorithm. Two of the best algorithms for solving ARE, namely the Schur vector approach and the eigen number approach are discussed in detail. The algorithm for $C(SI - A)^{-1}B$ function, polynomial multiplication, and state variable feedback design are developed. The detail of these algorithms are presented in the next chapter.

CHAPTER 4

IMPLEMENTATION

1. Introduction

In the previous chapter, MATLAB and the algorithm to solve ARE were chosen. The algorithms that are not available are developed during this study. The chapter first describes the process of incorporating the new routines into ICECAP. It then discusses how to incorporate MATLAB into ICECAP and follows by presenting the developed algorithms in detail. Finally, the implementation of the developed routines is discussed.

2. Incorporating MATLAB

In the early design stage, ICECAP's structure was designed to grow in a tree-like fashion so that new modules could be easily added. However, the main driver of the ICECAP, usually called the executive module, is not well implemented. For example, it consists of too many submodules. These submodules are contained in one big file. Therefore, whenever a change is made to a single module, all other modules must also be re-compiled. In addition to this, the executive interprets the user command and translates it to an option number. The option number is then passed to the FORTRAN modules which in turn execute this option number just as if it had been entered by the user using TOTAL. This implementation makes the program run

slower. Also, the executive module calls the VAX library routine. This ties the program itself to the host computer, which violates one of the requirement objectives.

Since it is not the purpose of this thesis to make a major modification to the executive module, eliminating the deficiencies discussed above are suggested for future students. This thesis, however, does employ a new way for incorporating the new modules. The subsection below discusses MATLAB's structure and the approach used in incorporating it into ICECAP.

Appendix A includes a railroad diagram which describes various syntactic quantities suach as command, expression, term, and factor. These quantities are used in performing matrix operations. The structure of the parser/interpreter presented in Reference [7] is repeated in this thesis for continuity. The structure of the parser/interpreter is similar to that of Wirth's compiler [44]. The interrelation of the primary subroutines is shown in the Figure 2. The detailed description of each subroutine can be found in Appendix B.

Subroutine "Parse" controls the interpretation of each statement. It calls subroutines that process the various syntactic quantities such as command, expression, term and factor (see Appendix A for the definition of these quantities). A fairly simple program stack mechanism allows

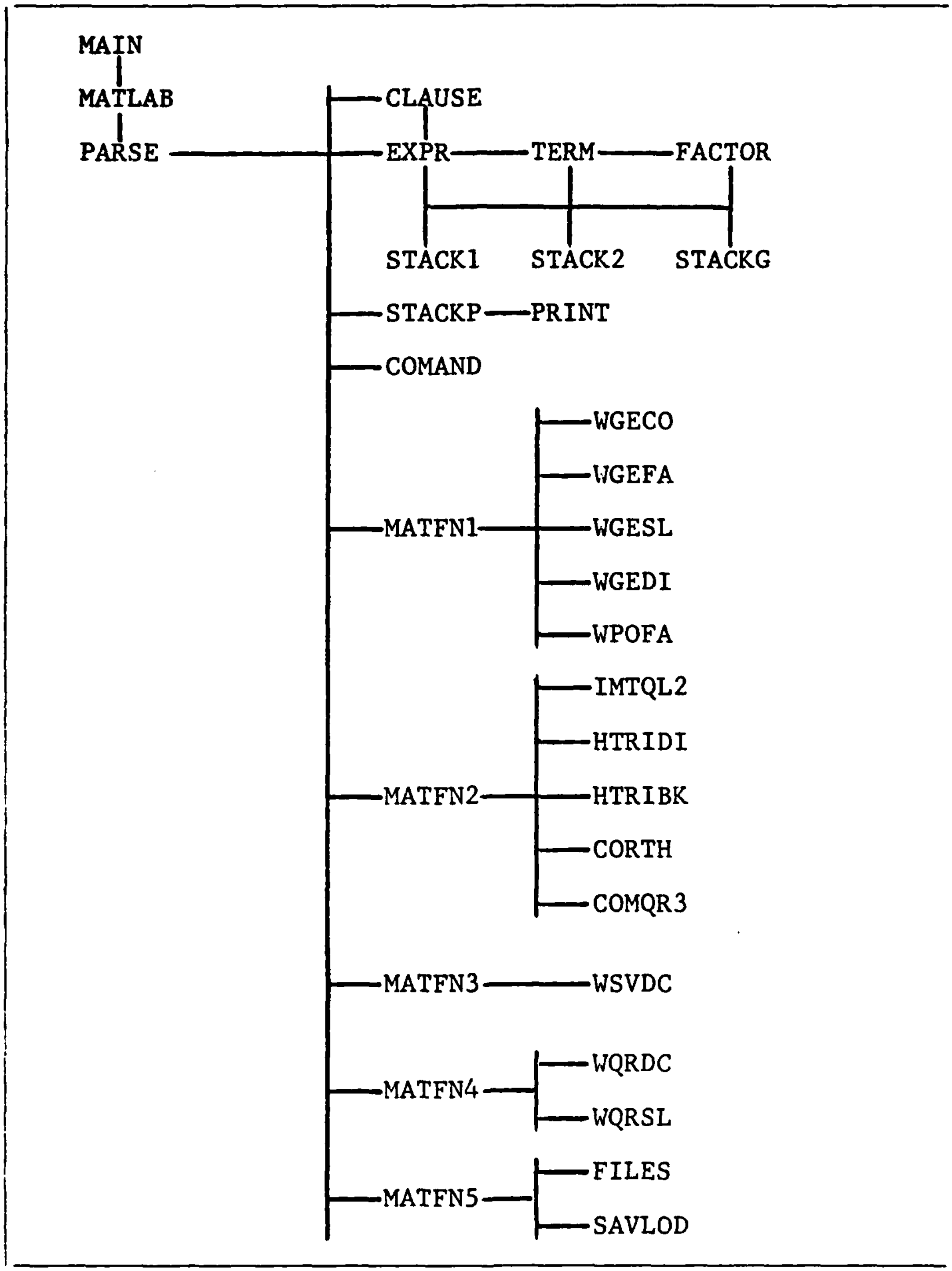


FIGURE. 2 MATLAB'S STRUCTURE

these subroutines to "call" each other recursively. The four stack subroutines, namely STACK1, STACK2, STACKP, and STACKG, manage the variable memory and perform elementary operations, such as matrix addition and subtraction. The four subroutines MATFN1 though MATFN4 are called whenever "serious" matrix computations are required. They are interface routines which in turn call the various LINPACK and EISPACK subroutines. MATFN5 primarily handles the I/O file operations. Two large double precision real single arrays, STKR and STKI, are used to store all the elements of matrices. Real numbers are stored in STKR and for complex numbers, the real parts are stored in STKR while the imaginary parts are stored in STKI. Four integer arrays, namely IDSTK, MSTK, NSTK, and LSTK are used to store the names, the row and column dimensions, and the pointers into the stacks, respectively. TOP and BOT integer variables are used to indicate whether the stacks are full or not. Figure 3 illustrates this storage scheme. The top portion of the stack is used for computation working space while the bottom portion is reserved for saved variables. The figure shows the situation after the line

A = <11, 12 ; 21, 22>, X = <3.14 ; SQRT(-1)>

has been entered to the MATLAB program (Notice that a semicolon is used to separate each row). The two variables, A and X, have dimensions 2 by 2 and 2 by 1 and so take up a

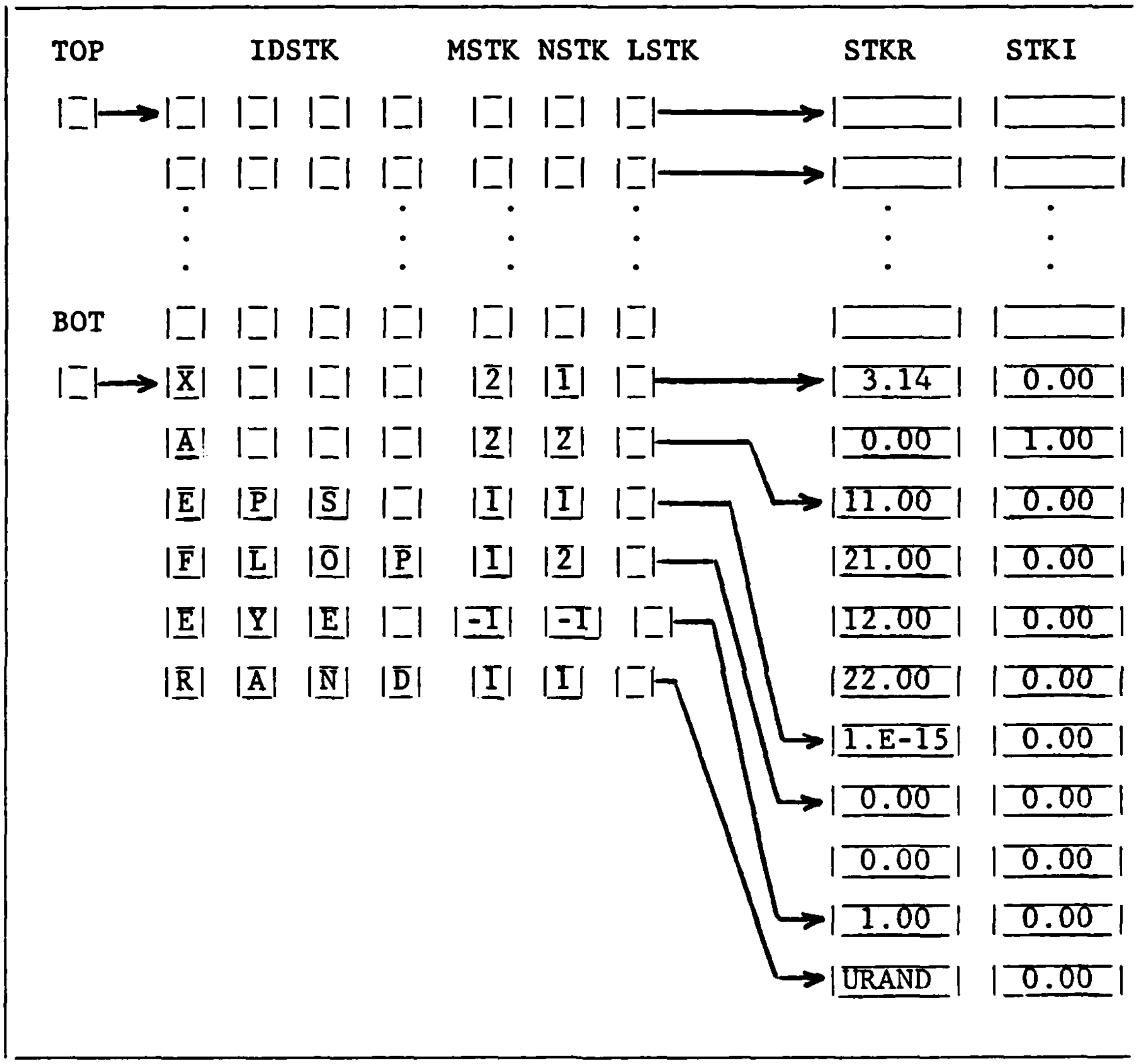


FIGURE. 3 MATLAB's DATA STRUCTURE

total of 6 locations. Notice that MATLAB accepts A and X by rows, but it stores them internally by columns. The subsequent statement involving A and X will result in temporary copies being made to the top of the stack for use

in the actual calculations. The four permanent names, EPS, FLOP, RAND, and EYE, occupy the last four positions of the variable stacks. RAND has dimension 1 by 1; its value is provided by a random number generator. EYE has dimension -1 by -1 to indicate that the actual dimensions must be determined later by context.

The modular structure of MATLAB (see Figure 2) makes it possible to implement it on ICECAP. The implementation is accomplished by using the syntax diagram for two specific commands, the DEFINE and DISPLAY commands. These two commands were originally chosen by Gembarowski [4]. The syntax diagram was developed as a tree-like structure wherein each module displays the choices that are appropriate for the command that has been partially formulated. For instance, in formulating the command DEFINE MATRIX, the Pascal procedure DEFINE calls Pascal procedure DEFINE_PROMPT, if necessary, in order to prompt the user as to valid choices for the second word. Once the valid command string is formed, the MATLAB parser routine is called directly and the user is in the MATLAB program which is considered to be a lower level of ICECAP. In this level, the user can utilize all MATLAB commands and all data entered in this level are stored in the MATLAB's data structure. The reason for this implementation is that the main menu is getting bigger as ICECAP is getting larger.

IMPLEMENTATION

There is no place to store large matrices, such as 50x50 matrices, in ICECAP data structure. Therefore, it is necessary to step down to a lower level.

The structure of MATLAB permits the user to define the names of variables as opposed to ones strictly defined by the program. This is very useful. For example, in multiplying two polynomials, the user can multiply them and save the result by simply giving it the new name. This eliminates the copy command (i.e. copy POLYA to POLYD option 66 in TOTAL). Additionally, and more importantly, the data will be allowed to pass from MATLAB's data structure to ICECAP's data structure or vice versa. This gives the user an ability to compute, for example, the open-loop transfer function in a lower level and to use this transfer function for other controller design in the upper level without retyping.

3. Implementing The Algorithm

3.1 Riccati Solver. It is not the purpose of this thesis to present the theory that supports the eigen number approach. The interested reader may refer to Reference [42]. This section mainly considers numerical issues such as algorithm implementation, timing, storage, and stability. To solve the optimal control, consider a dynamic system

$$\dot{X} = AX + Bu \quad ; \quad Y = CX \qquad (1)$$

and a cost function

IMPLEMENTATION

$$J = \int_0^\infty (X^T Q X + u^T R u)\, dt \qquad (2)$$

where Q and R are weighting matrices

The problem is to compute the feedback matrix, K, so that

$$u = -KX \qquad (3)$$

minimizes Eq (2) subject to the differential constraint of Eq (1).

K is given by

$$K = R^{-1} B^T P \qquad (4)$$

where P is the solution to the steady state Riccati Equation

$$A^T P + PA - PBR^{-1} B^T P + Q = 0 \qquad (5)$$

and the dimensions of matrices given by

A,P,Q all are n-by-n

B is n-by-r

R is r-by-r.

The eigen number approach is used to solve Eq (5). The algorithm is presented as follows :

1. Form

$$S = \begin{bmatrix} -A & D \\ -Q & A^T \end{bmatrix} \in R^{2nx2n}$$

where

$$D = BR^{-1} B^T$$

2. Compute the eigenvalues of S i.e. Solutions to

$$\left| \lambda I - S \right| = 0$$

3. Select the n eigenvalues with positive real part

4. Form a polynomial of order n using the eigenvalues selected in step 3; i.e.,

$$P_n(s) = s^n + a_{n-1} s^{n-1} + \dots + a_1 s + a_0$$

5. Form a 2nx2n matrix Z using the coefficients of the polynomial in step 4 and S in step 1; i.e.,

$$S^n + a_{n-1} S^{n-1} + a_{n-2} S^{n-2} + \dots + a_1 S + a_0 I = Z \in R^{2nx2n}$$

6. Partition Z as

$$Z = \begin{bmatrix} U_{nxn} & M_{nxn} \\ V_{nxn} & N_{nxn} \end{bmatrix}$$

7. Compute

$$P_1 = M^{-1} U$$

$$P_2 = N^{-1} V$$

$$P_3 = -V^{-1} U$$

$$P_4 = -N^{-1} M$$

where P_1, P_2, P_3, P_4 all are solutions to (5)

A unique solution of equation (5) for a positive definite P exists if the matrix Q is positive definite [34].

IMPLEMENTATION

The inverse of M, N, V may not exist when Eq (5) is known to be "ill-conditioned" [12]. There is no guarantee that one of those solutions computed in step 7 will be positive definite. However, from the results of testing of the algorithm, the positive definite solution is most likely obtained with selecting n eigenvalues having positive real part (step 3). If the positive definite solution is not obtained, step 3 must be repeated. That is selects another n eigenvalues and repeats step 4 thru 7. This process is repeated until the positive definite is obtained. However, the existance of a uniqueness of positive definite solution condition should be checked before computing the solutions.

Good estimates of the condition number of U, M, and N with respect to inversion are computed by the MATLAB linear equation software with estimates being inspected during computation. For an "ill-conditioned Riccati equation" U, M, and N may have no inverse. Methods for computing solutions for such cases are discussed further in Reference [48].

With respect to storage considerations the algorithm requires at least two 2nx2n arrays. The overall process is quite stable numerically (see Section 4.1.2 in Chapter 5) and as indicated above if U, M, and N are not invertible, it is still possible to compute the solutions.

Once the positive definite solution is obtained, the

feedback matrix, K, can be computed by using Eq (4). Then, the closed-loop matrix can be computed as

$$A_{cl} = A - BK \qquad (6)$$

where

A_{cl} is the closed-loop matrix

Finally, the closed-loop matrix can be used to compute the closed-loop transfer function as

$$G_{cl}(S) = C(SI - A_{cl})^{-1} B \qquad (7)$$

MRIC is coded to solve this optimal control problem.

3.2 The algorithm for $C(SI - A)^{-1} B$ provides the numerator and denominator polynomials of the transfer function for a single-input single-output system of Eq (1). In order to compute the numerator polynomial, it is necessary first to have available the coefficients of the characteristic polynomial, which is obtained by computing the determinant of (SI - A); i.e.,

$$\left| SI - A \right| = S^{n} + a_{n-1} S^{n-1} + a_{n-2} S^{n-2} + \ldots + a_{1} S + a_{0}$$

Once the denominator is obtained, numerator coefficients can be computed as follows :

$$b_n = CB$$

$$b_{n-1} = CAB + a_{n-1} CB$$

$$b_{n-2} = CA^2B + a_{n-1} CAB + a_{n-2} CB$$

$$\vdots \qquad \vdots \qquad \vdots \qquad \vdots$$

etc.

This algorithm is short and simple since a FORTRAN DO loop can generate b's. The subroutine TRFF is coded to compute this $C(SI - A)^{-1}B$ function.

3.3 Polynomial Operations. The addition and subtraction of polynomials are accomplished by calling subroutine STACK2. Multiplication is coded using the algorithm shown below :

$$P_1(X) = X^n + a_{n-1} X^{n-1} + a_{n-2} X^{n-2} + \dots + a_1 S + a_0$$

$$P_2(X) = X^n + b_{n-1} X^{n-1} + b_{n-2} X^{n-2} + \dots + b_1 S + b_0$$

$$\begin{bmatrix} 1 \\ a_{n-1} \\ a_{n-2} \\ \vdots \\ a_0 \end{bmatrix} \cdot \begin{bmatrix} 1 & b_{n-1} & b_{n-2} \dots & b_0 \end{bmatrix} = \begin{bmatrix} 1 & b_{n-1} & b_{n-2} & \dots & b_0 \\ a_{n-1} & a_{n-1}b_{n-1} & a_{n-1}b_{n-2} \dots & & a_{n-1} b_0 \\ a_{n-2} & a_{n-2}b_{n-1} & a_{n-2}b_{n-2} \dots & & a_{n-2}b_0 \\ \vdots & \vdots & \vdots & & \vdots \\ a_0 & a_0 b_{n-1} & a_0 b_{n-2} \dots & & a_0 b_0 \end{bmatrix}$$

$$P_1(X) \times P_2(X) =$$

$$X^n + (a_{n-1} + b_{n-1})X^{n-1} + (a_{n-2} + a_{n-1}b_{n-1} + b_{n-2})X^{n-2} + \ldots + a_0 b_0$$

The subroutine MPOLY is coded to perform this polynomial multiplication.

3.4 State Variable Feedback Design. Modern control theory introduces the concept of using all the system states to provide the desired improvement in system performance. The state variable feedback concept requires that all states be accessible in a physical system, but for most systems this requirement is not met; i.e., some of the states are inaccessible. A technique for handling systems with inaccessible states is presented in [31,34]. The state variable feedback design method presented in this section is based upon achieving a desired control ratio for a single-input single-output system; i.e., it is a pole placement technique.

As mentioned previously, this thesis investigates the state variable feedback design using phase variable representation. The main objective is to provide a state feedback gain corresponding to the desired closed-loop transfer function. Due to the ease of using phase variable representation, the state equation in physical variables is transformed to a phase variable form. The solution is

transformed back to physical variables for actual implementation on digital computer. The algorithm for this is shown below :

1. Given A, B, C matrices, the open loop transfer function, $G_{ol}(S)$ can be computed by

$$G_{ol}(S) = C(SI - A)^{-1}B = \frac{K_g(S^w + b_{n-1}S^{w-1} + \dots + b_1S + b_0)}{S^n + a_{n-1}S^{n-1} + \dots + a_1S + a_0}$$

$$= \frac{K_gN(S)}{Q(S)}$$

where K_g is an open loop forward gain and $w < n$

The desired closed-loop transfer, $G_{cl}(S)$ is given by

$$G_{cl}(S) = \frac{K_gN(S)}{Q_{cl}(S)}$$

where $Q_{cl}(S) = S^n + \hat{a}_{n-1}S^{n-1} + \dots + \hat{a}_1S + \hat{a}_0$

2. The system error is defined by

$$e(t) = r(t) - y(t)$$

For a zero steady state error, $e(t)_{ss} = 0$, the steady state ouput must be equal to the input; i.e.,

$$y(t)_{ss} = r(t)$$

thus, K_g can be computed as

$$K_g = \frac{\hat{a}_0}{b_0}$$

3.

$$K^*_n = \frac{\hat{a}_{n-1} - a_{n-1}}{K_g}$$

where K^*_n is a n-th state feedback gain in phase variable form.

4. Compute the transformation matrix, T, that needs phase variable form and the original form given in A, B, and C as :

$$\begin{aligned} T_n &= B \\ T_{n-1} &= AT_n + a_{n-1} B \\ T_{n-2} &= AT_{n-1} + a_{n-2} B \\ &\vdots \end{aligned}$$

etc.

$$T = \left[T_1 , T_2 , \ldots , T_n \right]$$

5.

$$K^T = K^{*T} T^{-1}$$

MODERN is coded to perform this state variable feedback design function.

IMPLEMENTATION

4. Incorporating the Developed Routines into ICECAP

4.1 Riccati Solver. To incorporate subroutine MRIC into ICECAP, a new command was added to the ICECAP second level menu. At this level, most of the commands are used to display results. For example, commands like DISPLAY SPECS, DISPLAY EQUATION are used to signal the FORTRAN modules to do computations and display results. With the same idea, DISPLAY RICCATI will signal MRIC to solve the Riccati equation. Additionally, if the solution is positive definite, MRIC will automatically compute the feedback matrix gain and the closed-loop matrix. The closed-loop matrix then can be used to compute the closed-loop transfer function by utilizing the subroutine TRFF. This implementation avoids adding too many commands into the main menu.

4.2 $C(SI - A)^{-1}B$ function. TRFF is implemented in ICECAP in a manner similar to that used for the Riccati solver routine. The command was chosen to be TRANSFER/F. This command is again added to the second level menu.

4.3 Polynomial Operations. Unlike MRIC and TRFF, polynomial operations are implemented directly in MATLAB. That is, all polynomial operations will take place in the MATLAB parser routines. This implementation was used to avoid adding another command to the ICECAP menu.

Additionally, the user is allowed to defined polynomial names and to perform polynomial operations sequencially without laborious separate steps.

4.4 State Variable Feedback Design. The MODERN subroutine is implemented in ICECAP in the same manner as MRIC and TRFF. The command was chosen to be MODERN. Again, two parts of command string, namely DISPLAY MODERN, must be formulated. MODERN is called directly and the user can then perform state feedback controller design using the MODERN subroutine.

At this point all development routines have been incorporated into ICECAP. Figure 4 shows a summary structure chart of the system's software construct.

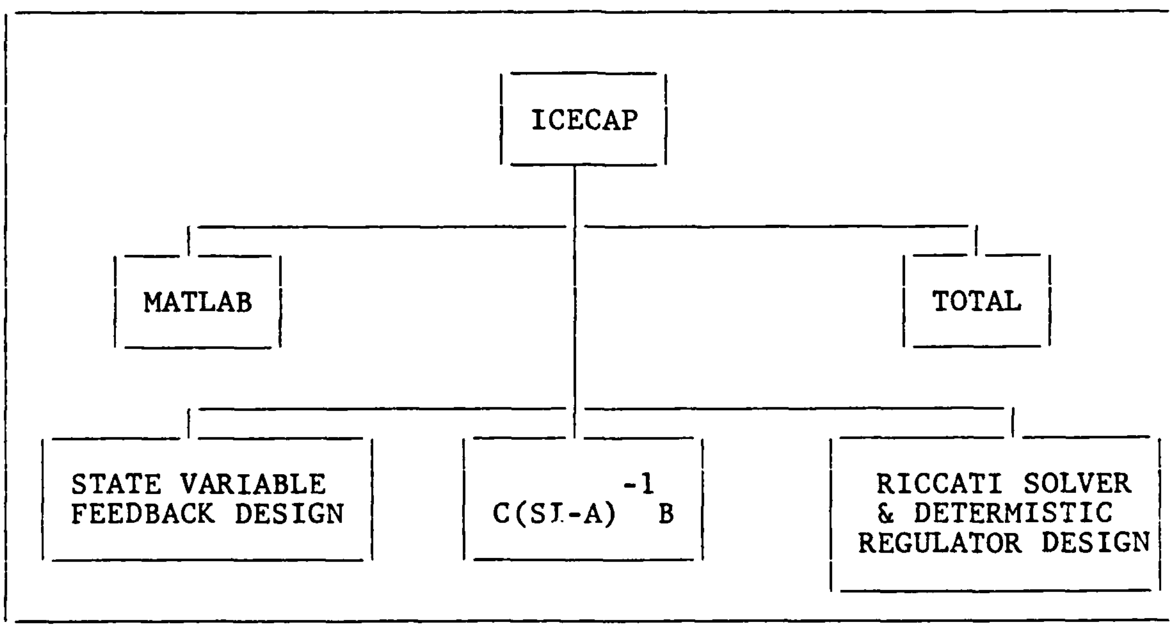


FIGURE 4 MODIFIED ICECAP'S STRUCTURE CHART

5. SUMMARY

In this chapter MATLAB's structure and the implementation of MATLAB in ICECAP were discussed. Details on specific commands of MATLAB were not presented. Those who are interested in MATLAB commands may refer to Appendix C and Reference [7]. Additionally, the algorithms of special routines were presented. Deeper understanding of the algorithms can be obtained from References [42,47]. Finally, implementing of the developed routines on ICECAP were presented. The details of each subroutine are shown in Appendix B, Subroutine Descriptions.

CHAPTER 5

TESTING AND RESULTS

1. Introduction

The testing of the MATLAB program and the development routines was accomplished concurrently with subroutine code development and implementation. As problems were discovered, they were analyzed for sources of error, and solutions were developed and implemented. These solutions were tested for problem corrections prior to continuing with development. In this manner, errors were not accumulated, thus making it much easier to accomplish final testing.

This chapter discusses the testing of the integrated part of ICECAP. The first section begins with testing philosophy. The next section deals with testing of the ICECAP-MATLAB portion which includes the initialization routine, command interpreter, commands and functions, and help module. The final section deals with testing and the validation of numerical results from the Riccati solver, $C(SI - A)^{-1}B$ function, polynomial operations, and state variable feedback design routines.

2. Testing Philosophy

Testing of a software system is the process of evaluating the performance of the program to insure that it is functioning as intended. Testing is not always a

distinct entity or process, rather it usually overlaps with other phases of development such as design and implementation. In this development, testing is accomplished to insure that the results obtained conform to the ICECAP design goals. Additionally, results using the Riccati solver, $C(SI - A)^{-1}B$ function, polynomial operations, and the state feedback design function must conform to the results obtained from other verified control system design program such as LQGLIB [39] using standard examples of Reference [7,12,34]. The test process used for this project is as follows :

* Verify syntax of routines
* Accomplish manual logic analysis to insure program as written approaches problem in proper way.
* Trace through code with known variables and known results to insure the coding produces the desired results.
* Display program variables at key spots to insure they agree with the precalculated number/results.

These methods provides an efficient means of detecting program errors and instituting corrections. The testing process and results are now discussed.

3. Testing ICECAP - MATLAB Portion.

3.1 Initialization. The testing of the initialization module is accomplished in two steps. First the program is run to verify that the initialization module sets various control flags as well as setting variables to their initial

values correctly. The results are satisfied when all initial values of variables and control flags are properly established. Second, the following areas are evaluated for the terminal initialization.

* Screen Clear - indicated by screen clearing before any other actions.

* Highlight and Nohighlight - indicated by proper part of menu being highlighted.

Again, the results are verified.

3.2 Command Interpreter. The command interpreter is tested in the following process :

* First line of instructions properly displayed

* Menu displayed

* Prompt displayed

* User cannot backspace past point where string has zero length.

* User can use either upper or lower case letters.

* Prompt redisplayed if command is incorrect.

* Error message displayed for incorrect command.

* User can use multiple commands.

Results : All of the above characteristics are confirmed 'good'.

3.3 Commands and Functions. The following commands and functions are tested.

- Commands :

    * HELP - Help options are displayed.

* KILL - Erase all variables, except EPS, FLOP, EYE, and RAND.

* EXIT & $ - Terminates MATLAB level and return the user to the system.

* LONG - The output of 15 digits accuracy is displayed.

* SHORT - The output of 4 digits accuracy is displayed.

* MENU - The commands and functions are displayed.

* DIR - All current variables are displayed.

* WHY - Various answers to any questions are displayed.

* CLEAR - The screen is clear.

Results : All of the above commands are confirmed 'good'.

- Functions. Using A, a real matrix of appropriate order and x, a number.

* INV(A) - inverse

* DET(A) - determinant

* COND(A) - condition number

* RCOND(A) - a measure of nearness to singularity; i.e, if A is well conditioned, RCOND(A) is near 1.0. If A is badly conditioned, RCOND(A) is near 0.0.

* EIG(A) - eigenvalues and eigenvectors

* SCHUR(A) - Schur triangular form

* HESS(A) - Hessenberg or tridiagonal form

* POLY(A) - characteristic polynomial

* SVD(A) - singular value decomposition

* PINV(A,EPS)	- pseudoinverse with optional tolerance
* RANK(A,EPS)	- matrix rank with optional tolerance
* LU(A)	- factors from Gaussian elimination
* CHOL(A)	- factor from Cholesky factorization
* QR(A)	- factors from Householder orthogonalization
* RREF(A)	- reduced row echelon form
* ORTH(A)	- orthogonal vectors spanning range of A
* EXP(x)	- E to the x
* LOG(x)	- natural logarithm
* SQRT(x)	- squart root
* SIN(x)	- sine
* COS(x)	- cosine
* ATAN(x)	- arctangent
* ROUND(A)	- round the elements to nearest integer
* ABS(A)	- absolute value of the elements
* REAL(A)	- real parts of the elements
* IMAG(A)	- imaginary parts of the elements
* CONJG(A)	- complex conjugate
* SUM(A)	- sum of the all elements
* PROD(A)	- product of all the elements
* DIAG(A)	- extract or create diagonal matrices
* TRIL(A)	- lower triangular part of A

* TRIU(A) - upper triangular part of A
* NORM(A,F) - norm with F = 1,2 or 'infinity'
* EYE(M,N) - a M by N matrix with 1's on the diagonal and zero elsewhere.
* RAND(M,N) - matrix of M by N with random entries
* ONES(M,N) - matrix of all ones
* MAGIC(N) - interesting test matrices
* HILBERT(N) - inverse Hilbert matrices
* ROOTS(C) - roots of polynomial with coefficients C
* DISPLAY(A,P) - print base P representation of A
* KRON(A,B) - Kronecker tensor product of A and B
* PLOT(X,Y) - plot Y as a function of X
* RAT(A) - find 'simple' rational approximation to A
* SAVE('file') - stores all the current variables in a file
* LOAD('file') - retrieves all the variables from a file
* PRINT('file',X) - prints X on a file
* DIARY('file') - makes a copy of the complete session

Results : All functions above perform properly.

3.4 Help Module. The help module provides on-line assistance to the user. It is tested with the following attributes :

* Program responds to request with correct information; indicated by verifying that proper Help information is displayed.

* User can abort Help by inputting a '$' at the end of Help information; indicated by return to command level.

Results : The above attributes are tested satisfactory.

4. Testing ICECAP - Development Modules.

4.1 Riccati Solver Module Testing. The Riccati solver testing scheme involves additional testing for numerical accuracy, speed, and numerical stability in addition to its logical evaluation of program flow and function. The following characteristics are tested in the Riccati solver module.

* Accepted input matrices' names correctly

* Numerical results are correct -- indicated by comparison with data from known source or substitute back to original equation.

* Help facility is working properly -- indicated by Help information is displayed correctly.

Results : The above characteristics test satisfactory.

4.1.1 Numerical accuracy and Speed testing. To achieve this testing scheme, the Riccati solver module and MATLAB program were transported to a better and bigger machine (better in a sense of machine speed and bigger in a sense of longer wordlength). The CDC Cyber 750 is used to serve this testing purpose. It also hosts the LQGLIB [39] software which solves the Riccati equation using the Schur vector approach. The results from both programs can be

compared both in speed and in numerical accuracy.

The example used to test is taken from a paper by A.J. Laub [12]. The Riccati equation is

$$A^T P + PA - PBR^{-1} B^T P + Q = 0$$

where all matrices are of order 9x9 and are given by

$$A = \begin{bmatrix} -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \end{bmatrix}$$

$$D = BR^{-1}B^T = \begin{bmatrix} 1 & & & & & & & & \\ & 0 & & & & & & \bigcirc & \\ & & 1 & & & & & & \\ & & & 0 & & & & & \\ & & & & 1 & & & & \\ & \bigcirc & & & & 0 & & & \\ & & & & & & 1 & & \\ & & & & & & & 0 & \\ & & & & & & & & 1 \end{bmatrix}$$

$$Q = \begin{bmatrix} 0 & & & & & & & & \\ & 10 & & & & & & \bigcirc & \\ & & 0 & & & & & & \\ & & & 10 & & & & & \\ & & & & 0 & & & & \\ & \bigcirc & & & & 10 & & & \\ & & & & & & 0 & & \\ & & & & & & & 10 & \\ & & & & & & & & 0 \end{bmatrix}$$

The 15 digits accracy of P from LQGLIB and MATLAB are compared in Figure 5. The solutions agree to at least 10 significant figures. Notice that only the first column of P is compared. The complete solution is available from the author.

LQGLIB using The Schur Vector Approach	MATLAB using The Eigen Number Approach
1.363020693809055	1.363020693808967
2.617215472388267	2.617215472388186
-.705427341233047	-.705427341232968
.936859701733908	.936859701733909
-.293666431891451	-.293666431891415
.477353860639198	.477353860639186
-.197375089533067	-.197375089533054
.211211652357913	.211211652358004
-.166551831151505	-.166551831151530

FIGURE 5 SOLUTIONS TO RICCATI EQUATION 9x9 SYSTEM

Substitution of a full 15 decimal place solution from LQGLIB into the Riccat equation gives a residual on the order of 10^{-14} while the residual for the solution from MATLAB is on the order of 10^{-26}. Both programs used FORTRAN5, but MATLAB used double precision while single precision is used in LQGLIB (Note that it is not practical to convert double precision to single precision for the entire program). For a speed comparison, LQGLIB is much faster than MATLAB, which we shall see in Table 1. The speed of MATLAB is very slow

since it has to multiply matrix of order 2nx2n n times, which is considered to be the most time consuming part of the algorithm (see Table 2). This is probably the worst aspect of algorithm.

TABLE 1

A TIME COMPARISON OF LQGLIB AND MATLAB

	MATRIX SIZE	CPU (sec.)
LQGLIB	2x2	0.007
	9x9	0.175
	20x20	1.444
MATLAB	2x2	0.069
	9x9	4.497
	20x20	62.440

TABLE 2

TIME ELAPSED IN EACH PROCESS RUN ON VAX 11/780

Matrix size	Form R and compute eigenvalues in sec	Sort eigenvalues and form polynomial in sec	Matrix Multiply in sec	$P=M^{-1}U$ in sec	Total time in min
9x9	7.20	0.05	5.64	0.32	00:13
20x20	7.21	0.15	135.75	1.37	02:30
30x30	23.39	0.27	778.11	4.15	14:45
40x40	54.95	0.51	2770.91	9.07	48:15
50x50	106.22	0.80	9402.13	18.95	2:25:52

4.1.2 Numerical Stability Testing. The example used to test is taken from the same source. All matrices are of order 20x20 and are given by

$$A = \begin{bmatrix} -2 & 1 & 0 & \cdots & 0 & 1 \\ 1 & & & & & 0 \\ 0 & & & & & 0 \\ \vdots & & \bigcirc & & & \vdots \\ 0 & & & & & 0 \\ 0 & & & & & 1 \\ 1 & 0 & 0 & \cdots & 0 \quad 1 & -2 \end{bmatrix}$$

$$D = BR^{-1}B^{T} = I$$

$$Q = I$$

The solutions were computed by MATLAB and checked against the solution from LQGLIB. The solutions agree to at least 12 significant figures. They are shown in Figure 6.

A higher order Riccati equation, for example 50^{th} order, was also tested on VAX 11/780 since the Cyber has a limitation on the allocated core memory. The solution was substituted into the Riccati equation. The residual is on the order of 10^{-7}. This is due to a 32 bits wordlength of the VAX, for a large system, the accuracy begins to loose. The numerical solutions are available from the author.

A comparison of the Schur vector approach and the eigen number approach is summarized in Table 3.

LQGLIB using The Schur Vector Approach	MATLAB using The Eigen Number Approach
p_{11} = .475050712641757	p_{11} = .475050712641716
p_{12} = .503886212983314	p_{12} = .503886212983235
.	.
.	.
.	.
p_{119} = .171194090783836	p_{119} = .171194090783858
p_{120} = .226250904062361	p_{120} = .226250904062364

FIGURE 6 SOLUTIONS TO RICCATI EQUATION 20x20 SYSTEM

TABLE 3

A COMPARISON OF

THE SCHUR APPROACH AND THE EIGEN NUMBER APPROACH

	Schur method	Eigen number method
Speed	fast	slow
Storage	at least two 2nx2n arrays	same
Reliability	handle up to 100 order	same
Solution	no guarantee of symmetry [12]	symmetric guarantee
# of solutions	produces one solution	produces all solutions

TESTING AND RESULTS

Since the Riccati solver module was also programmed to perform the optimal control function, the test for this particular part is conducted below :

Problem 15-2 on page 714 of Reference [34] is used. The problem statement is as follows :

$$\dot{X} = \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix} X + Bu \; ; \; B_1 = \begin{bmatrix} 10 \\ 0 \end{bmatrix} \; ; \; B_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$Y = \begin{bmatrix} 1 & 0 \end{bmatrix} X \; ; \; Q_a = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \; ; \; Q_b = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \; ; \; Z = 1$$

(a) Find the feedback coefficient matrix for this system for each matrix B. Obtain solutions for each matrix Q indicated. (b) For the matrix B_1, compare the time responses for a unit step input.

The testing procedures are :

1. DEFINE MATRIX i.e. enter A, B_1, B_2, Z, C, Q_a and Q_b
2. DISPLAY RICCATI i.e. enter the names of input matrices.
3. DISPLAY TRANSFER/F i.e. compute closed-loop transfer function
4. COPY OLTF CLTF i.e. closed-loop transfer function must be in CLTF
5. DISPLAY SPECS i.e. displayed fiqure of merit.
6. DISPLAY EQUATION i.e. display the output Y(t) with unit step input.

The solutions for part a) are

Using B_1, Q_a gives

$$P = \begin{bmatrix} 0.09850415 & 0.00742327 \\ 0.00742327 & 0.00155601 \end{bmatrix}$$

$$K = \begin{bmatrix} 0.98504158 \\ 0.07423273 \end{bmatrix}$$

Using B_2, Q_b gives

$$P = \begin{bmatrix} 0.77288396 & 0.09824936 \\ 0.09824936 & 0.19168385 \end{bmatrix}$$

K is the same as X

where P is the solution to Riccati equation
K is the feedback matrix gain

The solutions to part b) using B_1,Q_a are shown in Figure 7. The results agree to the solutions given on page 729 [34] to at least 6 significant figures.

4.2 $C(SI - A)^{-1}B$ Module Testing. To verify the validation of this module, the example taken from page 445 of Reference [34] is used. The problem statement is as follows:

$$\dot{X} = \begin{bmatrix} -2 & 0 \\ -1 & -1 \end{bmatrix} X + \begin{bmatrix} 1 \\ 1 \end{bmatrix} u$$

$$Y = \begin{bmatrix} 0 & 1 \end{bmatrix} X$$

Determine the transfer function Y(s)/U(s).

The test procedures are

```
              CLOSED-LOOP TRANSFER FUNCTION (CLTF)

                  CLK= ( CLNK/CLDK )=   10.0

                      CLTF (S) NUMERATOR

I        CLNPOLY(I)                         CLZERO(I)
1  (     1.000    )S** 1      (  -3.000    ) + J(  0.000E+00)
2  (     3.000    )                      CLNK=    10.00

                      CLTF (S) DENOMINATOR

I        CLDPOLY(I)                         CLPOLE(I)
1  (    1.000    )S** 2       (  -9.774    ) + J(  0.000E+00)
2  (    12.85    )S** 1       (  -3.076    ) + J(  0.000E+00)
3  (    30.07    )                       CLDK=   1.000


   CONTINUOUS TIME RESPONSE FOR CLTF(S)
   WITH STEP INPUT OF STRENGTH =   1.00000

  THE TIME FUNCTION IS
  F(T) =

           -1.0347    EXP(-9.7743     T)
           0.36921E-01EXP(-3.0761     T)
           0.99779    EXP(0.00000E+00T)

     CONTINUOUS TIME RESPONSE FOR CLTF(S)
     WITH STEP INPUT OF STRENGTH =    1.00000

        RISE TIME:          TR=   0.211244
        DUPLICATION TIME:   TD=   0.497606
        PEAK TIME:          TP=   0.670213
        SETTLING TIME:      TS=   0.354482
        PEAK VALUE:         MP=   1.001000
        FINAL VALUE:        FV=   0.997784
```

FIGURE 7. THE TIME RESPONSE OF UNIT STEP INPUT

1. DEFINE MATRIX i.e. enter A, B, C
2. DISPLAY TRANSFER/F i.e. compute a transfer function based upon A, B, C matrices

Note: The system will prompt with a message "ENTER YOUR A,B,C,D MATRICES :". D is a direct transmission matrix. For this example, there is no D matrix. The system will assume D equals 0 (its default value). The output is shown in Figure 8.

```
                    OPEN-LOOP TRANSFER FUNCTION (OLTF)

                     OLK = GAIN*(OLNK/OLDK)=     1.00

                              GAIN=   1.00

                          OLTF(S) NUMERATOR
I         OLNPOLY(I)                              OLZERO(I)
1   (     1.000     )S** 1         (  -1.000   ) + J(  0.000E+00)
2   (     1.000     )                        OLNK=     1.000

                        OLTF(S)  DENOMINATOR
I         OLDPOLY(I)                              OLPOLE(I)
1   (     1.000     )S** 2         (  -2.000   ) + J(  0.000E+00)
2   (     3.000     )S** 1         (  -1.000   ) + J(  0.000E+00)
3   (     2.000     )                        OLDK=     1.000
```

FIGURE 8. THE OPEN-LOOP TRANSFER FUNCTION

The result is exactly the same as the one given in the example [34] (cancel the zero and pole at -1.000).

4.3 Polynomial Operations Module Testing. This function was tested by using the procedure summarized below:

1. DISPLAY MATRIX i.e. go to MATLAB level

2. Enter the coefficients of polynomial from high to low.

$$X^2 + 2X + 1 \longrightarrow A = \langle 1 \quad 2 \quad 1 \rangle$$
$$X^2 + 4X + 4 \longrightarrow B = \langle 1 \quad 4 \quad 4 \rangle$$

3. Performs the operations

A*B, A+B, A-B, A**3, etc.

The results are as follows :

X = A*B : X = 1 6 13 12 4

Y = A+B : Y = 2 6 5

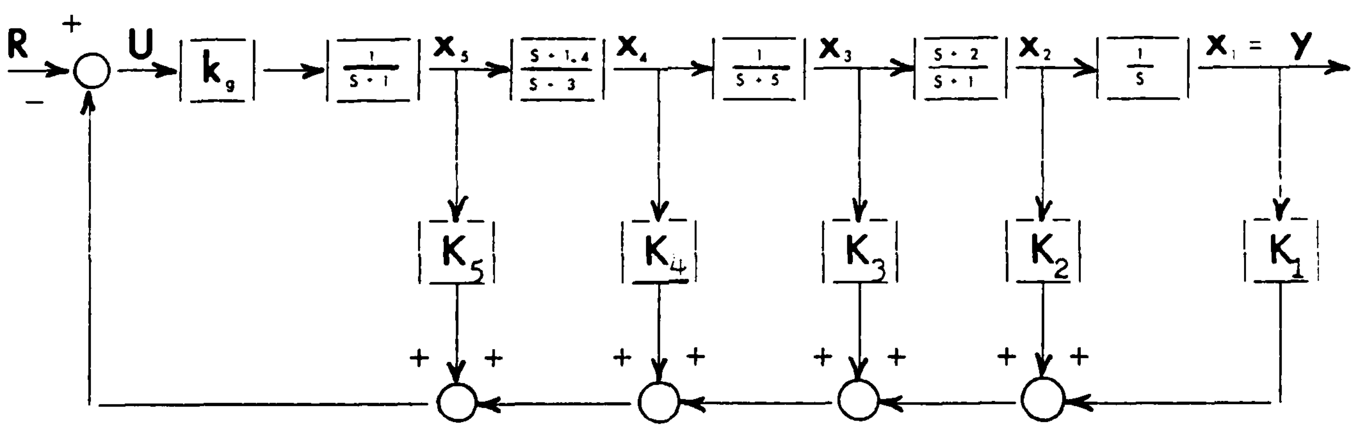
Z = A-B : Z = -2 -3

W = A**3 : W = 1 6 15 20 15 6 1

The results are checked against the results from TOTAL (using polynomial options, the polynomial operations on TOTAL have proved to be reliable polynomial operations). The results are confirmed 'good'.

4.4 State Feedback Design Module Testing. To test this module, the example on page 457 of Reference [34] is used. The problem statement is as follows:

For a system shown below



determine a feedback matrix gain K for a desired closed-loop transfer function below:

$$\frac{Y(S)}{R(S)} = \frac{100(S + 1.4)(S + 2)}{S + 75S + 360S + 710S + 704S + 280}$$

The testing procedures are

1. From the block diagram, determine A, B, C matrices
2. DEFINE MATRIX i.e. enter matrix A, then B,then C
3. DEFINE CLTF i.e. enter the desired closed-loop transfer function (CLTF) model
4. DISPLAY MODERN i.e. compute the feedback gain K

Note on step 1

One may determine A, B, C as follows:

$$\dot{X}_1 = X_2 \quad ; \quad \dot{X}_5 = -X_5 + U$$

$$\dot{X}_2 = -X_2 + \dot{X}_3 + 2X_3$$

$$\dot{X}_3 = -5X_3 + X_4$$

$$\dot{X}_4 = -3X_4 + \dot{X}_5 + 1.4X_5$$

From these equations, one may rewrite them as

$$\dot{X} = AX + Bu$$

$$Y = CX$$

That is

$$\dot{X} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & -1 & -3 & 1 & 0 \\ 0 & 0 & -5 & 1 & 0 \\ 0 & 0 & 0 & -3 & .4 \\ 0 & 0 & 0 & 0 & -1 \end{bmatrix} X + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} U$$

$$Y = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \end{bmatrix} X$$

Figure 9 shows the results.

Feedback matrix K from ICECAP	Feedback matrix K given on page 459 [34]
1.000000 1.000000 -2.441667 0.705208 -0.055208	1.00000 1.00000 -2.44167 0.70528 -0.05528

FIGURE 9 FEEDBACK MATRIX K

The results agree to at least 4 significant figures.

5. Summary

This chapter has discussed the ICECAP integrated part testing. The procedure for testing each module and the results of each test have been described. All the results are satisfactory. Thus, it is confirmed that, the ICECAP integrated parts are working properly.

CHAPTER 6

CONCLUSIONS AND RECOMMENDATIONS

1. Introduction

This chapter presents the conclusions reached during this investigation. The chapter also makes recommendations regarding the continuation of the efforts made in this thesis as well as other functions that need to be implemented.

2. Conclusions

The following conclusions were reached :

2.1 MATLAB was integrated into ICECAP. The integration of MATLAB into ICECAP is one of the most significant results of this effort. Its matrix operations are now part of ICECAP capabilities. Some of its routines were used to assist in computations for other functions. Other features such as I/O file operations can now be used to save, print, and recover the elements of matrices. On-line help assistance on matrix operations was also implemented properly. This on-line help is more effective than assistance that must be obtained from the manual.

2.2 The eigen number approach was coded and implemented on ICECAP. Although this approach is slow as compared to the Schur approach, it provides reliability and numerical accuracy as good as the Schur vector approach can (see Section 4 in Chapter 5). With all these capabilities,

ICECAP is now able to solve the Riccati equation and able to perform optimal control (regulator) design.

2.3 The routine to compute $C(SI - A)^{-1}B$ function was developed and implemented. ICECAP is now able to provide the transfer function for given A, B, C matrices. This function is very important since it can also be used to compute a closed-loop transfer function if a closed-loop matrix is known.

2.4 Polynomial Operations were enhanced. Although this function is currently available from TOTAL, its capability is cumbersome. For example, the name of polynomial is restrictively defined by the program. As opposed to TOTAL, ICECAP can perform the polynomial operations with much greater ease. For example, the user can define a polynomial name and perform multiple polynomial operations such as plus, minus, and multiplication sequencially without laborious separate step.

2.5 The routine to perform a state feedback design was developed and implemented. This routine was coded based on phase variable representation. As a result, ICECAP is now able to provide a state feedback matrix gain for a desired closed-loop transfer function.

3. Recommendations

The following recommendations are made :

3.1 Additional modern control functions should be

implemented. Although this thesis effort developed and implemented only the design by solving Riccati equation and the design via phase variable representation, other modern control functions yet to be implemented include :

* other state feedback design using canonical variable representation, observable variables representation, and generalized control canonical form
* linear regulator, observers (reduced order and full state feedback), and tracker using entire eigen structure assignment
* those listed in Section 3.1.2 of Chapter 2

3.2 Develop a routine that generates a generalized inverse. This routine can be used to compute the solution to Riccati equation in the case of uninvertible matrices [48].

3.3 Develop a routine that solves Riccati equation in discrete case as well as equivalent discrete models and sampled data form.

3.4 Modify ICER, the executive routine of ICECAP. As mentioned in Section 2 of Chapter 4, ICER was not well implemented. As the development continues, this routine must be modified.

3.5 Incorporate SOFE [49] & SOFEPL [50] programs. SOFE is used for generalized digital (Monte Carlo) simulation for optimal filter evaluation while SOFEPL is used as a postprocessor for SOFE that computes statistics and generates plots. The implementation of these two programs

on ICECAP will allow the user to perform stochastic estimation and control design.

3.6 Incorporate LQGLIB [39]. LQGLIB includes various computer routines that have applications in linear multivariable system studies. Although the source code for this program is not available because of copyright and contractual stipulations, the object code can be used to implement the program on ICECAP. When this is accomplished, ICECAP can be used as a linear multivariable systems design tool.

3.7 Improve computational efficiency of TOTAL's routines. Complete rework of numerical precision, numerical stability, and computational efficiency is required for high dimension.

4. Summary

Several conclusions were reached as a result of this thesis effort. This chapter has detailed these conclusions. This chapter also presents several recommendations regarding the continuation of ICECAP development. Recommendations on what functional capabilities that should be implemented on ICECAP are included.

BIBLIOGRAPHY

1. O'Brien, Frederick L. "A Consolidated Computer Program for Control System Design." MS Thesis. Wright-Patterson Air Force Base, Ohio. Air Force Institute of Technology, December 1976.

2. Larimer, Stanley J. "An Interactive Computer-Aided Design Program for Digital and Continuous Control System Analysis and Synthesis." MS Thesis. Wright-Patterson Air Force Base, Ohio. Air Force Institute of Technology, March 1978.

3. Logan, Glen T. "Development of An Interactive Computer-Aided Design Program for Digital and Continuous Control System Analysis and Synthesis." MS Thesis. Wright-Patterson Air Force Base, Ohio. Air Force Institute of Technology, March 1982.

4. Gembarowski, Charles J. "Development of An Interactive Control Engineering Computer Analysis Package (ICECAP) for Discrete and Continuous Systems." MS Thesis. Wright-Patterson Air Force Base, Ohio. Air Force Institute of Technology, December 1982.

5. Wilson, Robert E. "Continued Development of An Interactive Control Engineering Computer Analysis Package (ICECAP) for Discrete and Continuous Systems." MS Thesis. Wright-Patterson Air Force Base, Ohio. Air Force Institute of Technology, December 1983.

6. Smith, H.T. and T.R. Green. Human Interactive with Computers. New York, New York: Academic Press, 1980.

7. Moler, Cleve. "MATLAB Users' Guide." Department of Computer Science, University of New Mexico. May, 1981.

8. ...... "Solution of Linear-Quadratic Optimal Control Problem (OPTCON)." Air Force Institute of Technology Computer Program Library, 1974.

9. Fletcher, R. and Powell, M.J.D. "On The Modification of LDL' Facterizations," Mathematics of Computation, Vol.28 No.128 Oct. 1974, pp. 1067-1087.

BIBLIOGRAPHY

20. Frame, J.S. "Matrix Function and Application," IEEE Spectrum, Vol.1, No.6, June 1964, pp.

21. Patel, R.V. "On the Computation of Numerators of Transfer Functions of Linear Systems," IEEE Transactions On Automatic Control, Vol. AC-18, No.4, Aug. 1973, pp. 400-401.

22. Kleinman, D.L. "On an Iterative Technique for Riccati Equation Computations," IEEE Transactions on Automatic Control, Vol. AC-13, No.1, Feb. 1968, pp. 114-115.

23. Athans, M. and Falb, P.F. Optimal Control. New York: McGraw-Hill, 1966.

24. Marshall, C.P., III. Methods of Matrix Algebra. Academic Press, New York and London, 1965.

25. Hugh, G.C. Matrix with Applications. Appleton-Century-Crofts. Division of Meredith Corporation, New York, 1968.

26. Lancaster, P. Theory of Matrices, Academic Press, New York and London, 1969.

27. Cullen, C.G. Matrices and Linear Transformations. 2nd ed. Addison-Wesley Publishing Company, Massachusetts, 1972.

28. Householder, A.S. The Theory of Matrices in Numerical Analysis. Blaisdell, Waltham, Massachusetts, 1964.

29. Maybeck, P.S. Stochastic Models, Estimation, and Control. Vol.1 Academic Press, New York, 1982.

30. Maybeck, P.S. Stochastic Models, Estimation, and Control. Vol.2 Academic Press, New York, 1982.

31. Maybeck, P.S. Stochastics Models, Estimation, and Control. Vol.3 Academic Press, New York, 1982.

32. Klema, V.C. and Laub, A.J. "The Singular Value Decomposition. Its Computation and Some Applications," IEEE Transactions on Automatic Control. Vol. No. , April 1980. pp. 164-176.

BIBLIOGRAPHY

33. Houpis, C.H. and Lamont, G.B. Digital Control Systems (Theory, Hardware, Software). New York : McGraw-Hill, 1984.

34. D'Azzo, J.J. and Houpis, C.H. Linear Control System Analysis and Design Conventional and Modern. 2nd. ed. New York, New York : McGraw-Hill, 1981.

35. Bunch, J.R., Dongama, J.J., Moler, C.B. and Stewart, G.W. "Linpack Users' Guide," Siam Philadelphia, 1979.

36. Boyle, J.M., Dongarra, J.J., Klema, V.C., Garbow, B.S., Ikebe, Y., Moler, C.B. and Smith, B.T. "Matrix Eigensystem Routines-Eispack Guide," 2nd ed. Springer-Verlag, Berlin, Heidelberg, New York, 1976.

37. Melsa, J.L. and Schultz, D.G. Linear Control Systems. New York : McGraw-Hill, 1969.

38. Aseltine, J.A. Transform Method in Linear System Analysis. New York : McGraw-Hill, 1958.

39. Floyd, R.M. "LQGLIB User's Manual," Department of Electrical Engineering, Air Force Institute of Technology. May, 1984.

40. Walker, Robert, Charles Gregory, Jr, and Sunil Shah. "MATRIX_x : A Data Analysis, System Identification Control Design and Simulation Program," Abstract of Paper Awaiting publication.

41. Bangert, S.N. "CTRL-C," System Control Technology, INC., Advanced Technology Division, May 1984.

42. Jones, John, Jr. "Solutions of Certain Matrix Equations," Proceedings of the American Mathematical Society, 31: 333-339, 1972.

43. Potter, J.E. "Matrix Quadratic Solutions," SIAM J. Appl. Math, Vol.14 pp.496-501, 1966.

44. Plant, J.B. "Class Notebook for Automatic Control I, EE 481," Department of Electrical Engineering. Virginia Military Institute, Lexington, Virginia. August 1982.

45. Wirth, Niklaus. Algorithms + Data Structures = Programs. Prentice-Hall, 1976.

BIBLIOGRAPHY

46. Armold, A.T. "Further Development of An Interactive Control Engineering Computer Analysis Package (ICECAP) For Discrete and Continuous Systems," MS Thesis. Wright-Patterson Air Force Base, Ohio. Air Force Institute of Technology, December 1984.

47. Samuel D. Conte and Carl de Boor. Elementary Numerical Analysis. An Algorithmic Approach. 3rd Ed. McGraw-Hill Book Company, 1980.

48. Jones, John, Jr. "Handout on Generalized Inverse for Numerical Analysis, MA 508," Department of Mathematic. Air Force Institute of Technology, Ohio. December 1984.

49. Musick, S.H. "SOFE : A Generalized Digital Simulation for Optimal Filter Evaluation User's Manual," Reference System Branch, System Avionics Division, Wright-Patterson Air Force Base, Ohio. October 1980.

50 .... "SOFEPL : A Plotting Postprocessor for 'SOFE' User's Manual," University of Dayton Research Institute. Dayton, Ohio. November 1981.

51. Dale, N and Orshalick, D. Introduction to Pascal and Structured Design. D.C. Heath and Company. Lexington, Massachusetts / Toronto. 1983.

APPENDIX A

MATLAB's RAILROAD DIAGRAMS

1. Introduction

A formal description of the language acceptable to MATLAB was first described in Reference [7]. For the purpose of convenience, it is presented here again in a form of railroad diagrams. These railroad diagrams are read by following the arrow from entry to exit, one can construct a syntactically correct statement. For example, to define a name to a variable (matrix or polynomial or number), according to Figure 15, a name must has first a letter, then followed by more letters or digits. This process continues as long as one desires. For additional information see Reference [51] on page 31.

2. Railroad Diagrams

There are eleven non-terminal symbols in the language :

LINE, STATEMENT, CLAUSE, EXPRESSION,TERM,
FACTOR, NUMBER, INTEGER, NAME, COMMAND, TEXT.

The diagrams define each of the non-terminal symbols using the others and the terminal symbols :

```
Letter  --  A through Z,
Digit   --  0 through 9,
Char    --  ( ) ; : + - * / \ = . , < >
Quote   --  '
```

The railroad diagrams are presented as follows :

o CLAUSE	o FACTOR	o NUMBER	o TERM
o COMMAND	o INTEGER	o LINE	o TEXT
o EXPRESSION	o NAME	o STATEMENT	

RAILROAD DIAGRAM

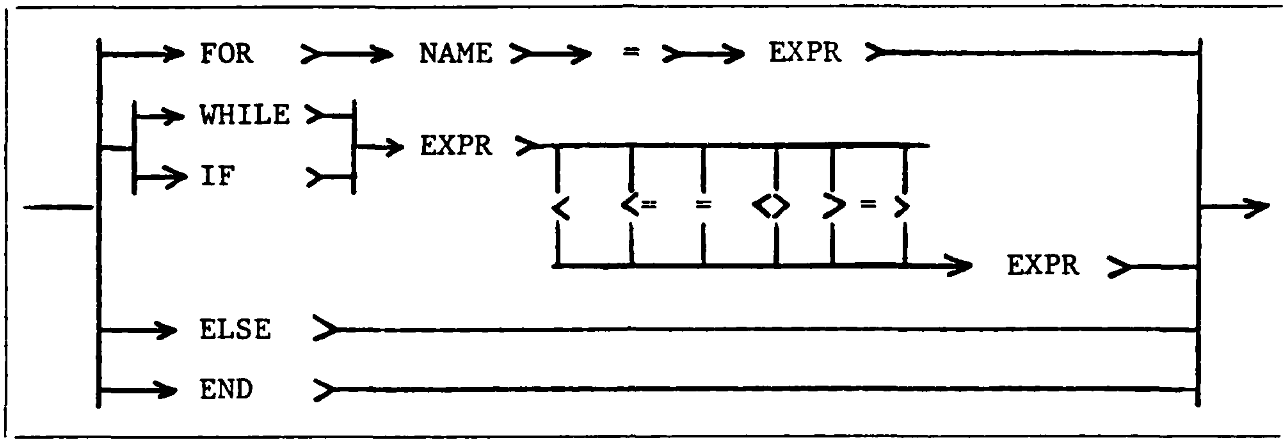


FIGURE 10 RAILROAD DIAGRAM FOR CLAUSE

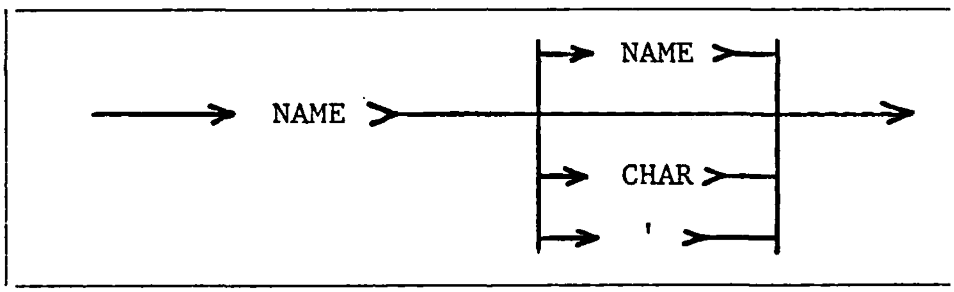


FIGURE 11 RAILROAD DIAGRAM FOR COMMAND

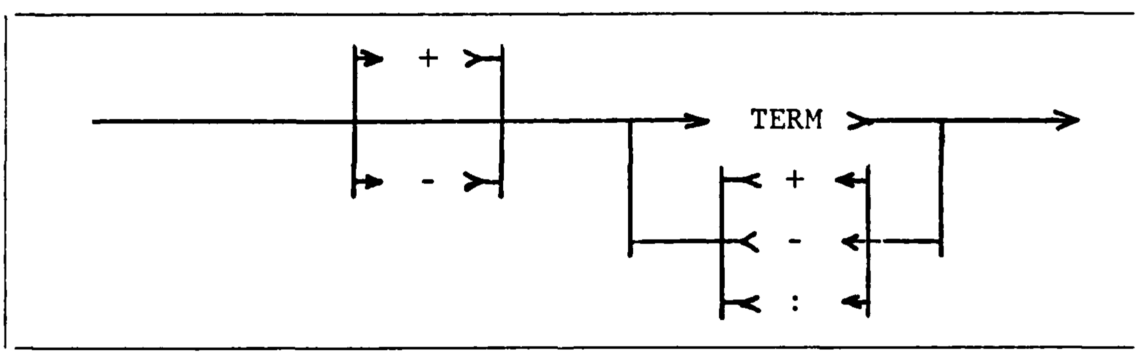


FIGURE 12 RAILROAD DIAGRAM FOR EXPRESSION

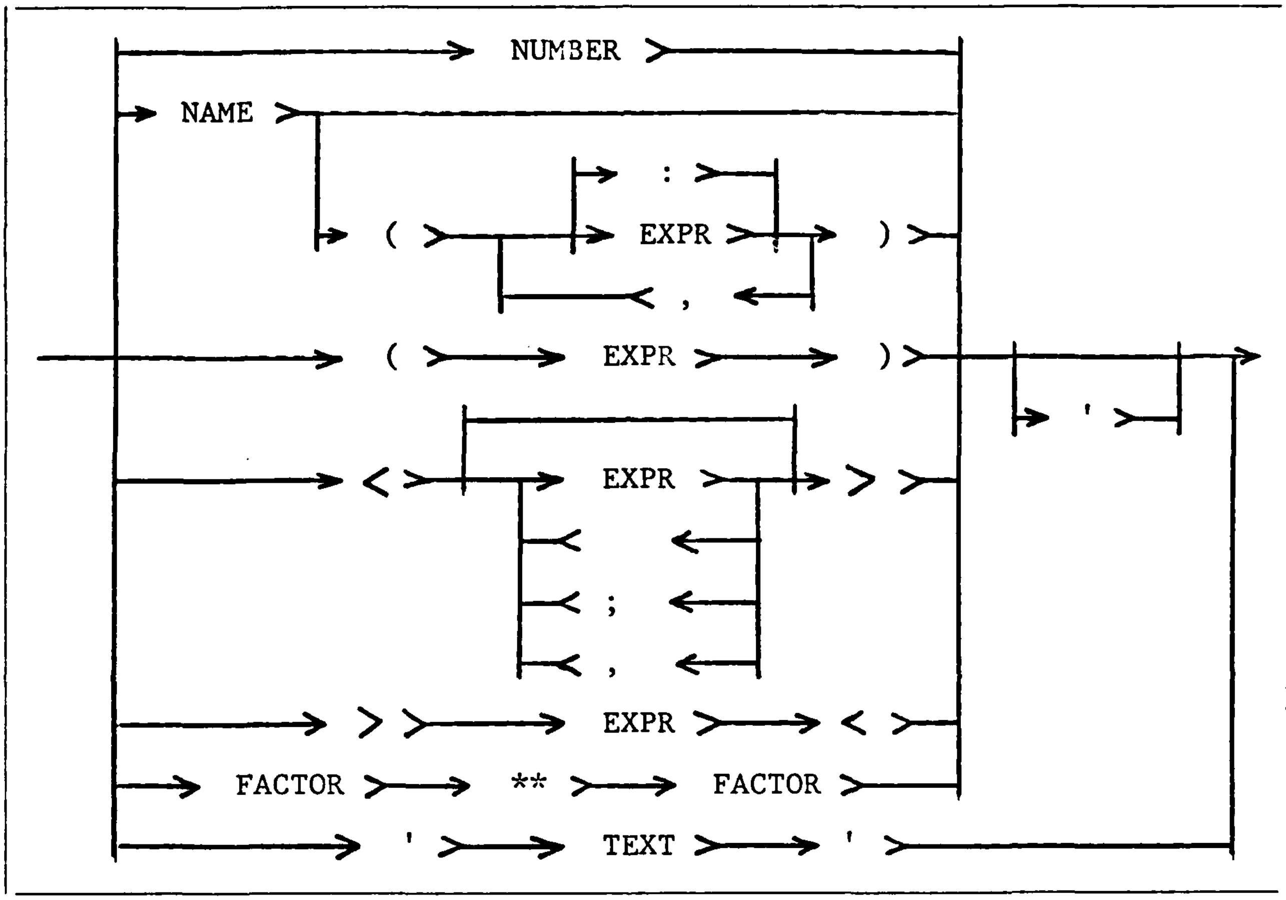


FIGURE 13 RAILROAD DIAGRAM FOR FACTOR

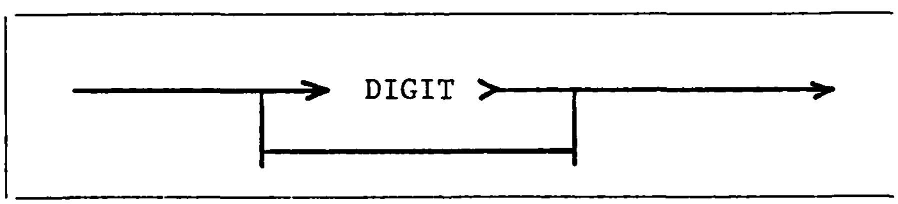


FIGURE 14 RAILROAD DIAGRAM FOR INTEGER

RAILROAD DIAGRAM

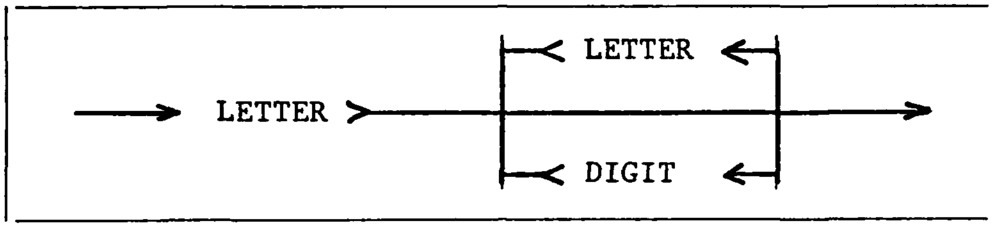


FIGURE 15 RAILROAD DIAGRAM FOR NAME

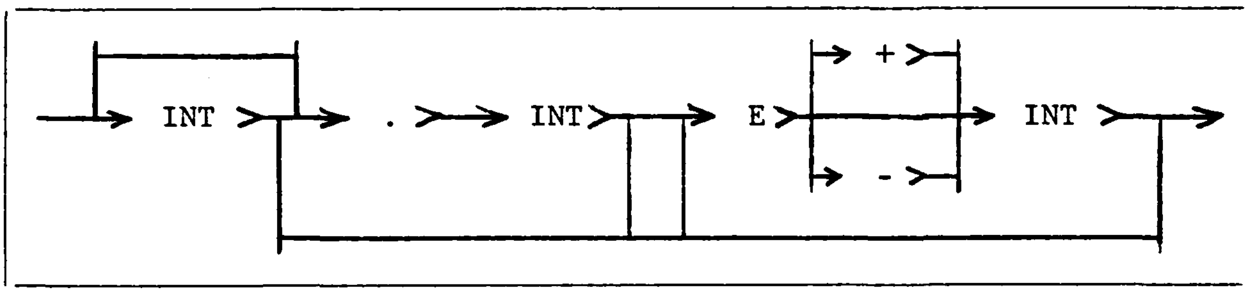


FIGURE 16 RAILROAD DIAGRAM FOR NUMBER

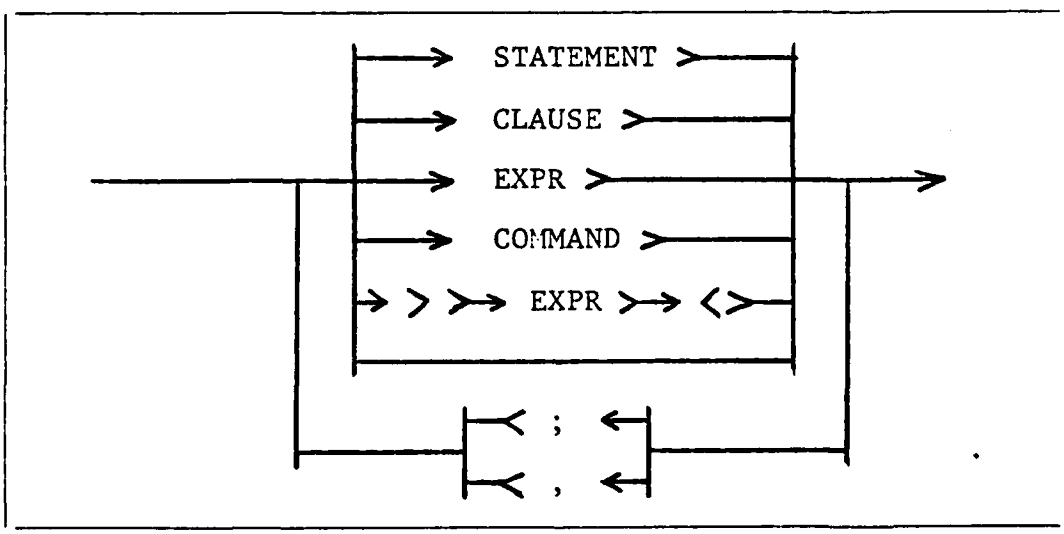


FIGURE 17 RAILROAD DIAGRAM FOR LINE

RAILROAD DIAGRAM

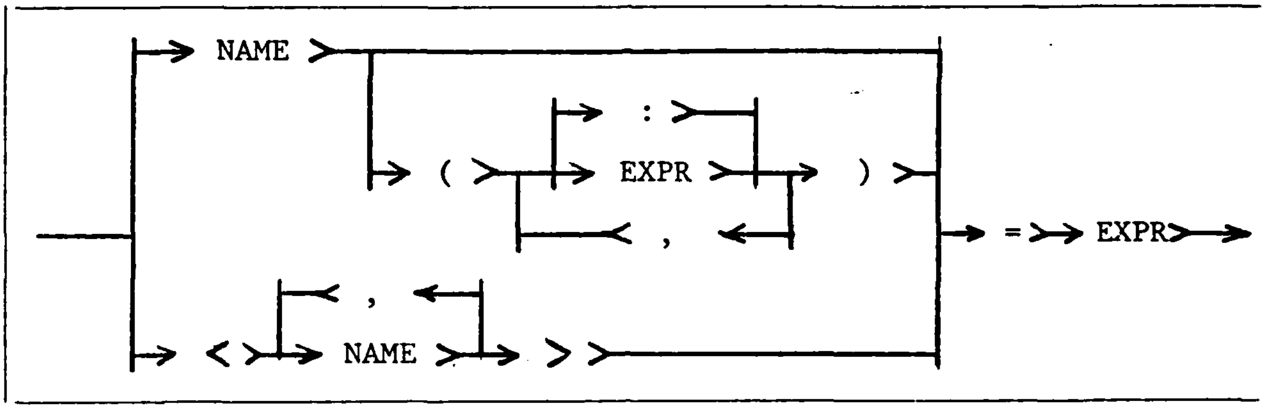


FIGURE 18 RAILROAD DIAGRAM FOR STATEMENT

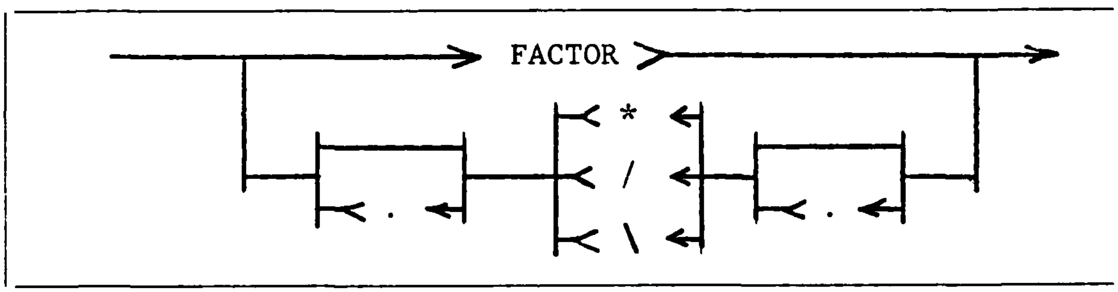


FIGURE 19 RAILROAD DIAGRAM FOR TERM

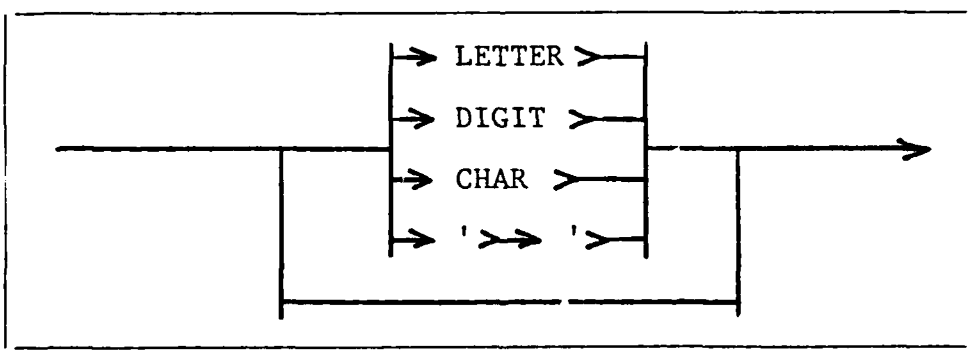


FIGURE 20 RAILROAD DIAGRAM FOR TEXT

APPENDIX B

SUBROUTINE DESCRIPTIONS

1. Introduction

This appendix gives a purpose, subroutines called, and a file name of each FORTRAN non-trivial subroutines used in MATLAB, MRIC, TRFF,and MODERN. All subroutines used double precision, single array for storing the elements of matrices. Certain routines from EISPACK and LINPACK software are well-documented in the source code; therefore, their descriptions are not included in this appendix.

2. List of Subroutine Descriptions

CLAUSE	FLOP	MATFN6	PROMPT	TRFF
CMULT	FORMR	MATLAB	PUTID	WPOLY
COMAND	FORMZ	MIXPOL	QROOT	XCHAR
CPOLY	FUNS	MODERN	QSAVE	
CROSS	GETCH	MPOLY	RPOLY	
CUT	GETLIN	MRIC	SAVLOD	
DESTOY	GETSYM	NEST	SORT	
DIAGON	GETVAL	NUM	STACK1	
EQID	MATFN1	OPTIMAL	STACK2	
ERROR	MATFN2	PARSE	STACKG	
EXPR	MATFN3	PLOT	STACKP	
FACTOR	MATFN4	PRINT	TERM	
FILES	MATFN5	PRNTID	TFORM	

SUBROUTINE DESCRIPTIONS

* Subroutine : CLAUSE

  Purpose : CLAUSE serves as a recursive routine. It handles FOR...NEXT, IF...THEN...ELSE...END, and WHILE features.

  Subroutines Called : GETSYM,ERROR,PUTID,WPOLY,STACKP, EXPR,PARSE

  File Name : CLAUSE.FOR

* Subroutine : CMULT

  Purpose : CMULT multiplies two complex numbers.

  Subroutines Called : None

  File Name : QCMULT.FOR

* Subroutine : COMAND

  Purpose : Sets up the MATLAB's command table. It also verifies whether the command is valid or not. If the command is valid, COMAND will process that command.

  Subroutines Called : ERROR,GETSYM,STACKP,FILES,PRNTID, FUNS

  File Name : COMAND.FOR

* Subroutine : CPOLY

  Purpose : Forms a polynomial of order n with n complex eigenvalues.

  Subroutines Called : CMULT, WPOLY

  File Name : QCPOLY.FOR

SUBROUTINE DESCRIPTIONS

* Subroutine : CROSS

  Purpose : Adds the elements of matrix i.e.

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} = \quad 1 \quad 6 \quad 15 \quad 14 \quad 9$$

  This subroutine is used in polynomial multiplication algorithm in Section 3.3 of Chapter 4.

  Subroutines Called : None

  File Name : QCROSS.FOR

* Subroutine : CUT

  Purpose : Partition a 2nx2n matrix as

$$Z = \begin{bmatrix} U_{nxn} & M_{nxn} \\ V_{nxn} & N_{nxn} \end{bmatrix}$$

  Subroutines Called : STACKG

  File Name : QCUT.FOR

* Subroutine : DESTOY

  Purpose : Erases the variables in the storage.

  Subroutines Called : STACKG,STACKP

  File Name : DESTROY.FOR

* Subroutine : DIAGON

  Purpose : Forms a diagonal matrix (nxn) with a given vector (nxl).

  Subroutines Called : None

  File Name : QDIAGON.FOR

SUBROUTINE DESCRIPTIONS

* Logical Function : EQID

  Purpose : EQID is used to check whether two given strings are the same or not.

  Subroutines Called : None

  File Name : LIB.FOR

* Subroutine : ERROR

  Purpose : Prints error messages.

  Subroutines Called : None

  File Name : ERROR.FOR

* Subroutine : EXPR

  Purpose : EXPR processes MATLAB's expression according to the railroad diagram for EXPR in Appendix A

  Subroutines Called : PUTID,GETSYM,ERROR,TERM,STACK1, STACK2

  File Name : EXPR.FOR

* Subroutine : FACTOR

  Purpose : FACTOR processes MATLAB's factor according to the railroad diagram for FACTOR in Appendix A.

  Subroutines Called : ERROR,GETSYM,GETCH,EXPR,STACK1, PUTID,FUNS,STACKG,MATFN's,STACK2

  File Name : FACTOR.FOR

* Subroutine : FILES

  Purpose : FILES ia a system dependent routine to allocate files.

  Subroutine Called : None

  File Name : FILES.FOR

SUBROUTINE DESCRIPTIONS

* Double Precision Function : FLOP

Purpose : FLOP is a system dependent double precision function. It counts and possibly chops each floating point operation.

Subroutine Called : None

File Name : FLOP.FOR

* Subroutine : FORMR

Purpose : Forms the matrix S based on A,Q,and D. where $D = BR^{-1}B^{T}$

$$S = \begin{bmatrix} -A & D \\ -Q & A^{T} \end{bmatrix}$$

Subroutines Called : STACK1

File Name : FORMR.FOR

* Subroutine : FORMZ

Purpose : FORMZ is a machine dependent routine which prints outputs with a Z format (see VAX/FORTRAN manual for Z format).

Subroutines Called : None

File Name : FORMZ.FOR

* Subroutine : FUNS

Purpose : FUNS sets up the MATLAB's functional table. It also verifies whether the function is valid or not. If the function is valid, FUNS set the two control variables namely, FIN and FUN which will be used to signal MATLAB subroutine to call functional routines, MATFN1 thru MATFN6.

Subroutine Called : PRNTID

File Name : FUNS.FOR

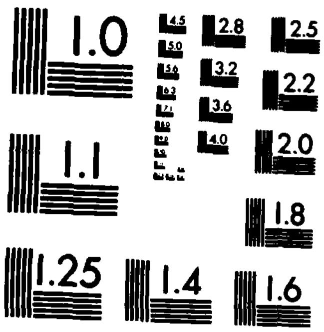
AD-A153 249 A MODERN CONTROL THEORY ENHANCEMENT TO AN INTERACTIVE CONTROL ENGINEERING..(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. C NARATHONG 2/2

UNCLASSIFIED DEC 84 AFIT/GE/ENG/84D-49 F/G 9/2 NL

END

FILMED

DTIC



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

SUBROUTINE DESCRIPTIONS

* Subroutine : GETCH

  Purpose : GETCH gets next character from the buffer.

  Subroutines Called : None

  File Name : GETCH.FOR

* Subroutine : GETLIN

  Purpose : GETLIN reads in the input and puts it in the buffer 80 characters at a time.

  Subroutines Called : XCHAR,GETCH,PUTID,FILES,EDIT

  File Name : GETLIN.FOR

* Subroutine : GETSYM

  Purpose : GETSYM primarily verifies each character in the buffer which contains 80 characters. This buffer was read previously in by subroutine GETLIN.

  Subroutines Called : GETCH,GETVAL,PRNTID

  File Name : GETSYM.FOR

* Subroutine : GETVAL

  Purpose : GETVAL forms a numerical value of each character in the buffer.

  Subroutines Called : GETCH

  File Name : GETVAL.FOR

* Subroutine : MATFN1

  Purpose : MATFN1 evaluates functions involved in Gaussian elimination.

  Subroutines Called : ERROR,WGECO,WGESL,RSET,WCOPY,WGEDI, WGEFA,WSWAP,HILBER,WSCAL

  File Name : MATFN1.FOR

SUBROUTINE DESCRIPTIONS

* Subroutine : MATFN2

  Purpose : MATFN2 evaluates elementary functions and functions involved in eigenvalues and eigenvectors.

  Subroutines Called : ERROR,WCOPY,WSET,HTRIDI,IMTQL2, HTRINK,CORTH,COMQR3,WLOQ,WMUL,WATAN, WSQRT,WSCAL,WAXPY,WDIV

  File Name : MATFN2.FOR

* Subroutine : MATFN3

  Purpose : MATFN3 evaluates functions involved in singular value decomposition.

  Subroutines Called : ERROR,WSVDC,WCOPY,WRSCAL

  File Name : MATFN3.FOR

* Subroutine : MATFN4

  Purpose : MATFN4 evaluates functions involved in QR decomposition in least squares sense.

  Subroutine Called : ERROR,STACK1,WCOPY,WSET,WQRDC,WQRSL, WSWAP

  File Name : MATFN4.FOR

* Subroutine : MATFN5

  Purpose : MATFN5 performs file handling and other I/O operations.

  Subroutines Called : ERROR,FILES,PRINT,PUTID,SAVLOD, STACKP,RSET,RAT,BASE,WCOPY,STACK1, PLOT

  File Name : MATFN5.FOR

SUBROUTINE DESCRIPTIONS

* Subroutine : MATFN6

  Purpose : MATFN6 evaluates utility functions such as MAGIC,KRONECKER PRODUCT,SIZE,EYE,RAND, etc.

  Subroutines Called : ERROR,WCOPY,WMUL,WDIV,USER,RSET, MAGIC,WSET

  File Name : MATFN6.FOR

* Subroutine : MATLAB

  Purpose : MATLAB is used to initialize all necessary control variables and flags.

  Subroutines Called : FILES,WSET,PUTID,PARSE,MATFN1 thru MATFN6.

  File Name : MATLAB.FOR

* Subroutine : MIXPOL

  Purpose : Forms a polynomial of order n with given real and complex eigenvalues.

  Subroutines Called : WPOLY,CPOLY,RPOLY,QROOT

  File Name : MIXPOL.FOR

* Subroutine : MODERN

  Purpose : Performs a state variable feedback design using phase variable representation using the algorithm presented in Section 3.4 of Chapter 4.

  Subroutines Called : TFORM,STACKG,QSAVE,NUM

  File Name : MODERN.FOR

* Subroutine : MPOLY

  Purpose : Performs polynomial multiplication using the algorithm presented in Section 3.3 of Chapter 4.

  Subroutines Called : WPOLY

  File Name : MPOLY.FOR

SUBROUTINE DESCRIPTIONS

* Subroutine : MRIC

Purpose : MRIC solves the continuous time algebraic matrix Riccati equation

$$A^T P + PA - PBR^{-1} B^T P + Q = 0$$

using the eigen number approach.

Subroutine Called : FORMR,SORT,RPOLY,CPOLY,MIXPOL,NEST, CUT,ANSWER1,ANSWER2,QSAVE,DESTOY, MATFN2

File Name : RICCATI.FOR

* Subroutine : NEST

Purpose : Multiplies a matrix polynomial using a nest multiply algorithm [47].

Subroutines Called : STACK2,STACKG

File Name : QNEST.FOR

* Subroutine : NUM

Purpose : Computes coefficients of a numerator of a transfer function using the algorithm presented in Section 3.2 of Chapter 4.

Subroutine Called : MATMUL,MATVEC,VECPRO

File Name : NUM.FOR

* Subroutine : OPTIMAL

Purpose : Computes a feedback matrix gain K using a positive definite solution to Riccati equation and a closed loop matrix.

Subroutines Called : STACK2,STACKP,STACKG

File Name : OPTIMAL.FOR

SUBROUTINE DESCRIPTIONS

* Subroutine : PARSE

  Purpose : PARSE controls the interpretation of each statement. It calls subroutines that process the various syntactic quantities such as command, expression, term, and factor.

  Subroutines Called : FILES,PROMPT,GETLIN,PUTID,GETSYM, COMAND,FUNS,ERROR,STACKP,CLAUSE, EXPR,TERM,FACTOR

  File Name : PARSE.FOR

* Subroutine : PLOT

  Purpose : PLOT is used to plot X versus Y on specified unit number.

  Subroutines Called : None

  File Name : PLOT.FOR

* Subroutine : PRINT

  Purpose : PRINT serves as a primary output routine.

  Subroutines Called : FILES,PRNTID

  File Name : PRINT.FOR

* Subroutine : PRNTID

  Purpose : PRNTID prints the variable names.

  Subroutines Called : None

  File Name : PRNTID.FOR

* Subroutine : PROMPT

  Purpose : PROMPT is used to issue MATLAB prompt with optional pause.

  Subroutines Called : None

  File Name : PROMPT.FOR

SUBROUTINE DESCRIPTIONS

* Subroutine : PUTID

  Purpose : PUTID is used to store the variable name onto the storage.

  Subroutines Called : None

  File Name : LIB.FOR

* Subroutine : QROOT

  Purpose : QROOT selects n positive real parts of eigenvalues.

  Subroutines Called : None

  File Name : QROOT.FOR

* Subroutine : QSAVE

  Purpose : Saves an output with a given name.

  Subroutines Called : GETLIN,GETSYM,STACKP

  File Name : QSAVE.FOR

* Subroutine : RPOLY

  Purpose : RPOLY forms a polynomial of order n with n real eigenvalues.

  Subroutines Called : DIAGON,MATFN2

  File Name : QRPOLY.FOR

* Subroutine : SAVLOD

  Purpose : SAVLOD is used for save and load data to and from the user disk.

  Subroutines Called : None

  File Name : SAVLOD.FOR

SUBROUTINE DESCRIPTIONS

* Subroutine : SORT

  Purpose : SORT rearranges the eigenvalues. It puts the largest eigenvalue in the top of the stack and the smallest in the bottom.

  Subroutines Called : None

  File Name : SORT.FOR

* Subroutine : STACK1

  Purpose : STACK1 performs unary operations and a transpose of a matrix since these operations are very simple. For a serious matrix computation, the LINPACK & EISPACK is used.

  Subroutines Called : WRSCAL,ERROR,WCOPY

  File Name : STACK1.FOR

* Subroutine : STACK2

  Purpose : STACK2 performs binary and ternary operations such as addition, subtraction, multiplication, etc.

  Subroutines Called : ERROR,WAXPY,WCOPY,WSCAL,WDIV,WMUL

  File Name : STACK2.FOR

* Subroutine : STACKG

  Purpose : STACKG is used to load data from the bottom of the stack to the top of the stack. This data will then be used in the actual computations.

  Subroutines Called : PUTID,ERROR,WCOPY

  File Name : STACKG.FOR

SUBROUTINE DESCRIPTIONS

* Subroutine : STACKP

Purpose : STACKP is used to put variables into stacks.

Subroutes Called : ERROR,FUNS,PUTID,WCOPY,WSET,PRINT

File Name : STACKP.FOR

* Subroutine : TERM

Purpose : TERM processes MATLAB's term according to the railroad diagram for TERM in Appendix A.

Subroutines Called : FACTOR,GETSYM,STACK2,ERROR,MATFN's

File Name : TERM.FOR

* Subroutine : TFORM

Purpose : Computes a transformation matrix which transforms a state equation from a physical variable form to a phase variable form.

Subroutines Called : STACKG,STACK2

File Name : TRANSFORM.FOR

* Subroutine : TRFF

Purpose : Performs a $C(SI - A)^{-1} B$ function.

Subroutines Called : STACKG,ERROR,MATFN2,NUM,VCOPY, MCOPY,COPYIER

File Name : TRANSFER.FOR

* Subroutine : WPOLY

Purpose : WPOLY multiplies a column vector of order (nxl) and a row vector (lxn) together.

Subroutines Called : CROSS

File Name : WPOLY.FOR

SUBROUTINE DESCRIPTIONS

* Subroutine : XCHAR

  Purpose : XCHAR is a system dependent routine to handle special characters.

  Subroutines Called : None

  File Name : XCHAR.FOR

APPENDIX C

ICECAP USER's MANUAL

1. Introduction

This appendix first presents the overview of ICECAP. Second, it provides all ICECAP features. Finally, it includes examples of the use of ICECAP to solve modern control problems. They were selected from example problems or problems contained in the Reference [34].

2. Overview of ICECAP

ICECAP (Interactive Control Engineer Computer Analysis Package) [4,5] is a computer-aided design program that provides the control systems engineer with a "designer's workbench". Based on its current version, ICECAP can be used as a design tool such as conventional control design (discrete & continuous), modern and optimal control design. It is designed to be a system that is easy to use and learn; that is, it accepts single line command words from the user, processes them immediately, and displays the results. ICECAP is based on the program "TOTAL" [2] and "MATLAB" [7]. TOTAL is an interactive software package for digital and continuous control system analysis and synthesis (developed at The Air Force Institute of Technology). MATLAB was originally developed by C. Moler at the University of New Mexico. It was written as a convenient tool for computations involving matrices. MATLAB provides access to the LINPACK [35] and EISPACK [36] software; these two

packages represent the state of the art in matrix computational methods. EISPACK contains routines for matrix eigenvalue computations while LINPACK provides subroutines for solving and analyzing simultaneous linear equations. The MATLAB program has been enhanced with control design functions to form a complete interactive computer-aided control system design package.

General numerical analysis primitives perform the solution of simultaneous linear equations, matrix inversion, eigensystem analysis, singular value decomposition, and other matrix decompositions. Other specialized primitives are provided for conventional and modern control design. These include root locus design, state feedback design, and optimal control design (via the algebraic Riccati equation). Both continuous and discrete systems are supported.

ICECAP is much more than just a control design program. It provides on-line help assistance and quick answers to common problems, making it enjoyable to use.

3. MATRIX ANALYSIS

At the heart of ICECAP is the ability to manipulate matrices. It has commands and syntax that allow easy and powerful manipulation of matrices. Five stacks are used to store all variables and data. Matrices are contained in the local MATLAB "workspace". To see what objects are in the workspace, the command DIR is used.

Matrices can be introduced into ICECAP in several different ways. The easiest for small matrices is to use an explicit list. The explicit list is surrounded by '<' and '>', and uses the semicolon ';' to indicate the ends of the rows. For example, the input line

```
A = <1 2 3;4 5 6;7 8 9>
```

results in the output

```
A     =

    1.    2.    3.
    4.    5.    6.
    7.    8.    9.
```

which is be saved for later use. The individual elements are separated by commas or blanks and can be any MATLAB expressions. For example,

```
X =  <-9.4, 1/3, 4*atan(1)>
```

results in

```
X    =

  -9.4000   0.3333   3.1416
```

The command such as LONG R, X' provides results in

```
X    =

  -9.400000000000000
   0.333333333333333
   3.141592653589793
```

and the command SHORT restores the original format.

Large matrices can be spread across several input lines, with the RETURN replacing the semicolons. The above matrix could also have been produced by

```
A =    1 2 3
       4 5 6
       7 8 9
```

Matrices are overwritten, if they are assigned new values. For example,

```
A = <1 2;3 4>
```

results in

```
A    =

  1.   2.
  3.   4.
```

which replaces the previous A matrix.

All computations are done using double precision real arithemetic. The user, however, has a choice to see the results in REAL, E, D, or Z format. ICECAP has a rich instruction set for general matrix analysis. Most of these matrix primitives originated with MATLAB.

3.1 MATRIX ARITHMETIC

ICECAP provides a set of operations that perform basic matrix arithmetic :

/ - matrix right division computed by Gaussian elimination

\ - matrix left division computed by Gaussian elimination

' - matrix transpose, quote to delimit character strings

+ - matrix addition

- - matrix subtraction

* - matrix or polynomial multiplication

** - raises matrices or polynomials to powers

For example, suppose for a vector X and a matrix A, (Note that " [>:" is the system prompt)

```
[> : X  = <-1.3 4/5 pi>;
[> : A  = <1 2 3;4 5 6;7 8 10>;
```

a vector B is computed as A*X,

```
[> : B  =  A*X
```

which results in

```
B    =

   9.7248
  17.6496
  28.7159
```

then the statement

```
[> : Y  =  A\B
```

solves the linear equations and results in

```
Y    =

 -1.3000
  0.8000
  3.1416
```

The inverse of a matrix may also be formed directly using the function INV. If A had been non-square, then the under- or over- determined equation would have been solved in a least squares sense.

3.2 ELEMENT-BY-ELEMENT OPERATIONS

ICECAP has a set of elementary functions that operate on matrices. The following functions perform element-by-element operations. Given that A ia a matrix and x is a number,

ABS(x) - absolute value or magnitude

CONJ(x) - complex conjugate

IMAG(x) - imaginary part

PROD(A) - product of all elements

REAL(A) - real part

ROUND(A) - round to nearest integer

SIZE(A) - row and column dimensions of a matrix

SUM(A) - sum of all elements of a matrix

3.3 BASIC PROPERTIES

Some basic properties of a matrix may be calculated with the following functions : (given that A is a matrix)

COND(A) - condition number in 2-norm

DET(A) - determinant

INV(A) - inverse

HILB(A) - inverse HILBERT matrices

KRON(A) - Kronecker tensor product of two given matrices

NORM(A) - singular values, 1-norm, infinity norm, and F-norm; i.e., SQRT(SUM(DIAG(A'*A)))

PINV(A) - pseudoinverse with optional tolerance

RANK(A) - rank of a matrix

RAT(A) - remove roundoff error

RCOND(A) - estimate of the condition of a matrix

TRIL(A) - lower triangular part of a matrix

TRIU(A) - upper triangular part of a matrix

3.4 TRANCENDENTAL MATRIX FUNCTIONS

A set functions calculate the trancendental matrices that are defined for square matrices. (given that A is a matrix.)

ATAN(A) - arctangent

COS(A) - cosine

EXP(A) - matrix exponential

LOG(A) - natural logarithm

SIN(A) - sine

SQRT(A) - square root

These functions are calculated using eigenvalues and eigenvectors. When A is a vector, however, these functions are calculated on an element by element basis. If A is neither a vector, nor square, these functions give an error message.

3.5 DECOMPOSITIONS AND FACTORIZATIONS

Some matrix decompositions and factorizations may be calculated with the following commands : (given that A is a matrix)

CHOL(A) - Cholesky factorrization

EIG(A) - eigenvalues and eigenvectors

HESS(A) - Hessenberg form

LU(A) - factors from Gaussian elimination

ORTH(A) - orthogonalization

QR(A) - orthogonal-triangular decomposition

SHUR(A) - Schur decomposition

RREF(A) - reduced row echelon form of a rectangular matrix

SVD(A) - singular value decomposition

3.6 OTHER FUNCTIONS

ICECAP also provides some useful functions that help the user to generate often used matrices.

EYE - identity matrix

ONES - matrix of all ones

MAGIC - interesting test matrices. MAGIC(N) is an N by N matrix constructed from the integers 1 through N**2 with equal row and column sums.

3.7 POLYNOMIAL OPERATIONS

ICECAP provides several primitives for polynomial manipulations :

* & ** - multiplication and raise powers of polynomial

POLY - characteristic polynomial

ROOT - polynomial root

Polynomials are represented in ICECAP as row vectos containing the coefficients ordered by descending powers. For example, the characteristic equation of the matrix

```
A    =

     1.   2.   3.
     4.   5.   6.
     9.  10.  11.
```

is found by using POLY and typing

```
[> : P = POLY(A)
```

```
P    =

  1.0000
-17.0000
-24.0000
  0.0000
```

The roots of this equation (eigenvalues of matrix A) are found using ROOT.

```
[> : R = ROOT(P)

 R    =

  18.3107
  -1.3107
   0.0000
```

These may be reassembled into a polynomial using POLY.

```
[> :  PP = POLY(R)

    PP    =

      1.0000
    -17.0000
    -24.0000
      0.0000
```

Polynomial multiplication may be accomplished using "*". If A and B are polynomials, then Y = A*B calculates the polynomial product. For example, typing

```
[> : A = <1 2 1>;

[> : B = <1 2>;

[> : Y = A*B
```

which yields the polynomial product

```
Y   =

  1.   4.   5.   2.
```

Raising the power of polynomial can be done easily by

typing

```
[> : A = <1 1>;
[> : X = A**3
```

yields the polynomial product

```
X    =
   1.   3.   3.   1.
```

3.8 I/O FILES HANDLING

ICECAP provides a very useful file handling for saving and loading data.

DISP(T)	- print the text stored in T
SAVE('files')	- stores all the current variables in a file
LOAD('files')	- retrieves all the variables from a file
PRINT('FILE',X)	- print X on a file
DIARY('file')	- makes a copy of the complete ICECAP session

4. CONTROL DESIGN AND ANALYSIS

ICECAP provides a powerful environment for the analysis and design of control systems. Many analysis and design tasks are easily performed using a single application of a primitive function. A rich instruction set of primitive functions is available. Other design and analysis tasks can be solved by a short series of commands. The commands are interactively typed in by the user.

ICECAP is primarily concerned with linear systems that can be represented either in state-space form or in

polynomial notation as a Laplace transfer function for continuous time, or as a Z-transform transfer function for discrete time.

4.1 MODERN CONTROL

A system of LTI differential equations can always be expressed as a set of first-order matrix differential equations :

$$\dot{x} = Ax + Bu$$

$$y = Cx + Du$$

where u is the control input vector, x is a vector of state variables, and y is the output vector.

An equivalent representation for a single input-single output system is the Laplace transform representation :

$$\frac{Y(S)}{U(S)} = C\,(SI - A)^{-1}\,B + D$$

A system can be converted from state-space to Laplace transfer function by typing DISPLY TRANSFER/F and from there on the user can use ICECAP commands like DISPLAY, SPECS, LOCUS, EQUATION, etc. as desired.

ICECAP also provides other modern control design techniques, such as state feedback design, and optimal control design via the algebraic Riccati equation. Using a command like DISPLAY MODERN or DISPLAY RICCATI, the user can obtain the feedback matrix, and the solution to Riccati equation and feedback matrix associated with that positive

definite solution, respectively. Especially in the RICCATI solver, if the solution is positive definite, ICECAP also provides a closed loop matrix automatically. From this point on, the user can use the closed loop matrix to obtain a closed loop transfer function by typing DISPLAY TRANSFER/F.

4.2 CLASSICAL DESIGN (Continuous and Discrete System)

Since ICECAP builds upon the powerful routines of TOTAL, it retains the TOTAL's power as a classical design tool. Here are some useful ICECAP commands which can be used to perform control system design and analysis.

- HELP On line help is provided. Type HELP followed by...

  -- CHANGE (modifies the numerator or denominator constant, TSAMP and planes of analysis)

  -- COPY (copies source to destination)

  -- DEFINE (inputs Matrices and Transfer function)

  -- DELETE (Removes a pole or zero of a transfer function)

  -- DISPLAY (executes various functions)

  -- FORM (forms OLTF CLTF using GTF&HTF)

  -- INITIAL (explains abbreviations of commands)

  -- INSERT (adds a pole or zero to a transfer function)

  -- MATRIX (explains matrix functions)

  -- PRINT (prints data to answer file)

-- SYSTEM (explains genearl information about ICECAP)

-- TEACH (example of continuous time problems)

-- TFORM (for discrete transformation)

-- TURN (turn switches on or off)

- PRINT Print data to file(s). Type PRINT followed by...

-- EQUATION (invert Laplace transform)

-- GAIN (forward loop gain)

-- LISTING/F (listing of frequency responses)

-- LISTING/T (listing of time responses)

-- LOCUS AUTOSCALE (root locus with autoscale)

-- LOCUS MAGNIFY (magnifies root locus)

-- LOCUS SHRINK (reduces root locus)

-- LOCUS ZOOM (magnifies root locus at a particular point)

-- LOCUS/BRANCH (branch for each locus)

-- LOCUS/GAIN (locus with various gains)

-- LOCUS/ZETA (locus with various zetas)

-- PFE (partial fraction expansion)

-- SCAN/MAG (scanning magnitude)

-- SCAN/PHASE (scanning phase)

-- SPECS (figure of merit)

- RECOVER (load data from files)

- TURN

-- ANSWER (ON/OFF) (write data to file ANSWER)

-- DECIBELS (ON/OFF) (decibels mode)

-- GRID (ON/OFF) (grid on or off)

-- HERTZ (ON/OFF) (Hertz mode)

-- MAINMENU (ON/OFF) (mainmenu level)

- UPDATE save data to file MEMORY

- COPY source to destination (i.e. GTF to HTF etc.)

- DEFINE primary used for input data; various options are...

  -- MATRIX (enables matrix command)

  -- GAIN (modifies forward loop gain)

  -- CLTF POLY (input CLTF in poly form)

  -- OLTF FACT (input OLTF in factored form)

- DISPLAY primary output at user terminal, various options are...

  -- EQUATION (time response of CLTF)

  -- GAIN (forward loop gain)

  -- LISTING/F (listing of frequency responses)

  -- LISTING/T (listing of time responses)

  -- LOCUS AUTOSCALE (root locus with autoscale)

  -- LOCUS MAGNIFY (magnifies root locus)

  -- LOCUS SHRINK (reduces root locus)

  -- LOCUS ZOOM (magnifies root locus at a particular point)

  -- LOCUS/BRANCH (branch for each locus)

  -- LOCUS/GAIN (locus with various gain)

  -- LOCUS/ZETA (locus with various zeta)

-- MATRIX (same as DEFINE MATRIX)

-- MODERN (performs state variable feedback design)

-- OLTF, CLTF, HTF or GTF

-- PFE (partial fraction expansion)

-- RESPONSE/F (plots frequency response of transfer function)

-- RESPONSE/T (plots time response of a transfer function)

-- RICCATI (solve the continuous time ARE)

-- SCAN/MAG (scanning magnitude)

-- SCAN/PHASE (scanning phase)

-- SPECS (figures of merit)

-- SWITCHES (shows switch settings)

-- TRANSFER/F (performs $C(SI - A)^{-1} B$ function)

- FORM primary used for block diagram manipulation

  -- CLTF using GTF and HTF

  -- CLTF using OLTF

5. Command Language Definition

This section presents the ICECAP command language definitions in flow chart form. These definitions unambiguously define the ICECAP command language. Definitions are provided in alphabetical order. The standards used to develop the command language diagrams are also provided. Finally, an exhausted listing of every legal ICECAP command is presented along with the allowable

abbreviation in each case.

5.1 List of Command Language Definitions

A list of the described command language definitions appears below:

o CHANGE	o DISPLAY	o PRINT
o COPY	o FORM	o TURN
o DEFINE	o HELP	o TFORM
o DELETE	o INSERT	

5.2 Command Language Definition Standards

* All diagrams are to be read from left to right.

* Bracketed terms indicate choices. Only one choice per bracket is allowed.

* A lower case command word indicates that the feature has not yet been implemented in the language.

* The full spelling of each command word is used in each case. It is understood that the abbreviations described in section 5.3 are also valid.

* Also, the carriage return and the dollar sign are valid choices at any point in the diagram. The carriage return causes the system to prompt the user regarding the choices for the next allowable word. A dollar sign aborts the present commmand string.

* In addition at least one blank must separate the words in the command string.

* The blanks in some of the brackets are there only to give the diagram balance.

* Words that need no object in order to be complete commands are not shown. To date, this includes the commands STOP, UPDATE, and RECOVER.

The following figures describe the sequence of command words that form an ICECAP command.

```
[ CHANGE ]   [ CLDK  ]
             [ CLNK  ]
             [ GDK   ]
             [ GNK   ]
             [ HDK   ]
             [       ]
             [ HNK   ]
             [ OLDK  ]
             [ OLNK  ]
             [ PLANE ]
             [ TSAMP ]
```

FIGURE 21. COMMAND LANGUAGE DEFINITION FOR CHANGE

```
[ COPY ]   [ CLTF ]   [ CLTF ]
           [ GTF  ]   [ GTF  ]
           [      ]   [      ]
           [ HTF  ]   [ HTF  ]
           [ OLTF ]   [ OLTF ]
```

Figure 22. COMMAND LANGUAGE DEFINITION FOR COPY

$$\begin{bmatrix} \text{DEFINE} \end{bmatrix} \begin{bmatrix} \text{GAIN} \\ \text{MATRIX} \end{bmatrix}$$

$$\begin{bmatrix} \text{DEFINE} \end{bmatrix} \begin{bmatrix} \text{CLTF} \\ \text{GTF} \\ \text{HTF} \\ \text{OLTF} \end{bmatrix} \begin{bmatrix} \text{FACT} \\ \text{POLY} \end{bmatrix}$$

FIGURE 23. COMMAND LANGUAGE DEFINITION FOR DEFINE

$$\begin{bmatrix} \text{DELETE} \end{bmatrix} \begin{bmatrix} \text{CLTF} \\ \text{GTF} \\ \text{HTF} \\ \text{OLTF} \end{bmatrix} \begin{bmatrix} \text{ZERO} \\ \text{POLE} \end{bmatrix}$$

FIGURE 24. COMMAND LANGUAGE DEFINITION FOR DELETE

```
            CLTF
            EQUATION
            GAIN
            GTF
            HTF
            LISTING/F
            LISTING/T
            LOCUS/BRANCH
            LOCUS/GAIN
            LOCUS/ZETA
            MATRIX
DISPLAY
            MODERN
            OLTF
            PFE
            RESPONSE/F
            RESPONSE/T
            RICCATI
            SCAN/MAG
            SCAN/PHASE
            SPECS
            SWITCHES
            TRANSFER/F

                         AUTOSCALE
                         MAGNIFY
DISPLAY     LOCUS
                         SHRINK
                         ZOOM

                         CLTF
                         OLTF
DISPLAY     ROOT
            POLY         GTF
                         HTF
```

FIGURE 25. COMMAND LANGUAGE DEFINITION FOR DISPLAY

FORM

OLTF
CLTF USING GTF AND HTF
CLTF USING OLTF

FIGURE 26. COMMAND LANGUAGE DEFINITION FOR FORM

HELP

CHANGE
COPY
DEFINE
DELETE
DISPLAY
FORM
INITIAL
INSERT
MATRIX
PRINT
SYSTEM
TEACH
TFORM
TURN

FIGURE 27. COMMAND LANGUAGE DEFINITION FOR HELP

INSERT

CLTF
GTF
HTF
OLTF

ZERO
POLE

FIGURE 28. COMMAND LANGUAGE DEFINITION FOR INSERT

```
[ PRINT ]   [ CLTF          ]
            [ EQUATION      ]
            [ GAIN          ]
            [ GTF           ]
            [ HTF           ]
            [ LISTING/F     ]
            [ LISTING/T     ]
            [ LOCUS/BRANCH  ]
            [ LOCUS/GAIN    ]
            [ LOCUS/ZETA    ]
            [ OLTF          ]
            [ PFE           ]
            [ RESPONSE/F    ]
            [ RESPONSE/T    ]
            [ SCAN/MAG      ]
            [ SCAN/PHASE    ]
            [ SPECS         ]

[ PRINT ]   [ LOCUS ]   [ AUTOSCALE ]
                        [ MAGNIFY   ]
                        [           ]
                        [ SHRINK    ]
                        [ ZOOM      ]

[ PRINT ]   [ ROOT ]    [ CLTF ]
            [ POLY ]    [ OLTF ]
                        [      ]
                        [ GTF  ]
                        [ HTF  ]
```

FIGURE 29. COMMAND LANGUAGE DEFINITION FOR PRINT

TFORM	CLTF GTF HTF OLTF	SW SWP SZ WPS WPZ WS WZ ZW ZWP	BAKDIF BILINEAR IMPULSE TUSTIN MULTIPLE

FIGURE 30. COMMAND LANGUAGE DEFINITION FOR TFORM

TURN	ANSWER CANCEL CLOSED DECIBELS GRID HERTZ MAINMENU	ON OFF

FIGURE 31. COMMAND LANGUAGE DEFINITION FOR TURN

5.3 ICECAP Command

The following list contains every valid ICECAP command defined as of the conclusion of this investigation. The accepted abbrevaition is listed along side the command.

COMMAND	ABBREVIATION
CHANGE (numerator or denominator gain)	
CHANGE CLNK	CHA CLN
CHANGE PLANE	CHA PLA
CHANGE TSAMP	CHA TSA
COPY (source) (destinstion)	
COPY CLTF OLTF	COP CLT OLT
COPY GTF HTF	COP GTF HTF
DEFINE GAIN	DEF GAI
DEFINE INPUT	DEF INP
DEFINE (function) (fact/poly)	
DEFINE CLTF POLY	DEF CLT POL
DEFINE OLTF FACT	DEF OLT FAC
DEFINE MATRIX	DEF MAT
DELETE (function) (POLE or ZERO)	
DELETE HTF ZERO	DEL HTF ZER
DISPLAY (function)	
DISPLAY OLTF	DIS OLT
DISPLAY EQUATION	DIS EQU
DISPLAY GAIN	DIA GAI
DISPLAY LISTING/F	DIS L/F
DISPLAY LISTING/T	DIS L/T
DISPLAY LOCUS AUTOSCALE	DIS LOC AUT
DISPLAY LOCUS MAGNIFY	DIS LOC MAG
DISPLAY LOCUS SHRINK	DIS LOC SHR
DISPLAY LOCUS ZOOM	DIS LOC ZOO
DISPLAY LOCUS/BRANCH	DIS L/B
DISPLAY LOCUS/GAIN	DIS L/G
DISPLAY LOCUS/ZETA	DIS L/Z
DISPLAY MATRIX	DIS MAT
DISPLAY MODERN	DIS MOD

COMMAND	ABBREVIATION
DISPLAY PFE	DIS PFE
DISPLAY POLY (function)	
DISPLAY POLY GTF	DIS POLY GTF
DISPLAY RESPONSE/F	DIS R/F
DISPLAY RESPONSE/T	DIS R/T
DISPLAY ROOT (function)	
DISPLAY ROOT CLTF	DIS ROO CLT
DISPLAY SCAN/MAG	DIS S/M
DISPLAY SCAN/PHASE	DIS S/P
DISPLAY SPECS	DIS SPE
DISPLAY SWITCHES	DIS SWI
DISPLAY TRANSFER/F	DIS T/F
FORM OLTF	FOR OLT
FORM CLTF USING GTF AND HTF	FOR CLT USI GTF AND HTF
FORM CLTF USING OLTF	FOR CLT USI OLT
HELP CHANGE	HEL CHA
HELP COPY	HEL COP
HELP DEFINE	HEL DEF
HELP DELETE	HEL DEL
HELP DISPLAY	HEL DIS
HELP FORM	HEL FOR
HELP INITIAL	HEL INI
HELP INSERT	HEL INS
HELP MATRIX	HEL MAT
HELP PRINT	HEL PRI
HELP SYSTEM	HEL SYSTEM
HELP TEACH	HEL TEA
HELP TFORM	HEL TFO
HELP TURN	HEL TUR
INSERT (function) (POLE or ZERO)	
INSERT HTF ZERO	INS HTF ZER
PRINT EQUATION	PRI EQU
PRINT GAIN	PRI GAI
PRINT LISTING/F	PRI L/F
PRINT LISTING/T	PRI L/T
PRINT LOCUS AUTOSCALE	PRI LOC AUT
PRINT LOCUS MAGNIFY	PRI LOC MAG
PRINT LOCUS SHRINK	PRI LOC SHR
PRINT LOCUS ZOOM	PRI LOC ZOO
PRINT LOCUS/BRANCH	PRI L/B

COMMAND	ABBREVIATION
PRINT LOCUS/GAIN	PRI L/G
PRINT LOCUS/ZETA	PRI L/Z
PRINT PFE	PRI PFE
DISPLAY POLY (function)	
DISPLAY POLY GTF	DIS POLY GTF
PRINT RESPONSE/F	PRI R/F
PRINT RESPONSE/T	PRI R/T
PRINT ROOT (function)	
PRINT ROOT CLTF	DIS ROO CLT
PRINT SCAN/MAG	PRI S/M
PRINT SCAN/PHASE	PRI S/P
PRINT SPECS	PRI SPE
RECOVER	REC
STOP	STO
TFORM (function) ("from"plane"to"plane) (method)	
TFORM CLTF SZ TUSTIN	TFO CLT SZ TUS
TURN ANSWER (ON/OFF)	TUR ANS (ON/OFF)
TURN CANCEL (ON/OFF)	TUR CAN (ON/OFF)
TURN CLOSED (ON/OFF)	TUR CLO (ON/OFF)
TURN DECIBELS (ON/OFF)	TUR DEC (ON/OFF)
TURN GRID (ON/OFF)	TUR GRI (ON/OFF)
TURN HERTZ (ON/OFF)	TUR HER (ON/OFF)
TURN MAINMENU (ON/OFF)	TUR MAI (ON/OFF)
UPDATE	UPD

5.4 Discrete commands -- The discrete command consist of four words. The first word is TFORM. This informs the computer that a transformation is desired. The second word is the name of the transfer function. Presently, ICECAP has four transfer functions, CLTF , OLTF, GTF, and HTF. The third word is used to inform the computer the "existing" domain and the "desination" domain e.g. SZ where S is the "existing" domain and Z is the "destination" domain. The last word of the command string is the method. There are four methods of transformations i.e. Impulse, Tustin, Backward Difference, and Bilinear. The following are some typical discrete commands for ICECAP.

TFORM CLTF SZ IMPULSE Which is read: transform CLTF from the S-domain to the Z-domain via the Impulse method.

The inverse of the above is TFORM CLTF ZS IMPULSE.

Also, if one desires to perform a transformation from the S-domain to the W'-domain, one may use the word MULTIPLE as the fourth word of the command string. This will save time. For example, to transform a function from the S-domain to the W-domain requires two transformation. First from the S-domain to the Z-domain and the from the Z-domain to the W-domain. One way to do this is as follows:

TFORM GTF SZ IMPULSE then TFORM GTF ZW BILINEAR

Using MULTIPLE as the fourth word will do the same in one command i.e.

TFORM GTF SW MULTIPLE.

The following is a list of a few items to remember when performing a discrete analysis:

1. If the transfer function is in a domain other than the S-domain which is the default domain, then the sample time must be entered prior to performing the analysis. For example, if one wants to look at the discrete time response, enter CHANGE TSAMP. If, one wants the transfer function to be displayed with the correct parameters enter
CHANGE PLANE and ICECAP will prompt with a menu of desired planes.

2. Use the Impulse method to transform a transfer function with a ZOH or Padea approximation of a ZOH. If the transfer function contains a pole at the origin, then the ZOH will not work because the present algorithms do not allow repeated roots. An alternate way to solve this problem is to set the pole of the transfer function near zero i.e. .000001.

6. Example Problems.

Example 1. This example illustrates the use of function $C(SI - A)^{-1} B$ using example presented in Section 4-16 page 138 [34].

Problem : Determine the transfer functions and draw a block diagram for the two-input two-output system represented by

$$\dot{X} = \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix} X + \begin{bmatrix} 1 & 1 \\ 0 & -2 \end{bmatrix} U \; ; \quad Y = \begin{bmatrix} 0 & -2 \\ 1 & 0 \end{bmatrix} X$$

```
ICECAP> DEFINE MATRIX
[>: A = <0 1;-2 -3>,B1 = <1 0>;B1 = B1',B2 = <1 -2>;B2 = B2'
A    =
            0.   1.
           -2.  -3.
B1   =

            1.
            0.
B2   =

            1.
           -2.
[>: C1 = <0 -2>,C2 = <1 0>
C1   =

            0.  -2.
C2   =

            1.   0.
[>: $
ICECAP> DISPLAY TRANSFER/F
```

```
ENTER YOUR A,B,C,D MATRICES : A,B1,C1,D

                    OPEN-LOOP TRANSFER FUNCTION  (OLTF)

                     OLK = GAIN*(OLNK/OLDK)  =  4.000

                           GAIN=     1.000

                         OLTF(S) NUMERATOR
I      OLNPOLY(I)                                OLZERO(I)
1   (  1.000     )                              OLNK=   4.000

                         OLTF(S) DENOMINATOR
I      OLDPOLY(I)                                OLPOLE(I)
1   (  1.000     )S** 2          (   -2.000    ) + J(   0.0000E+00)
2   (  3.000     )S** 1          (   -1.000    ) + J(   0.0000E+00)
3   (  2.000     )                              OLDK=   1.000

ENTER YOUR A,B,C,D MATRICES : A,B1,C2,D

                    OPEN-LOOP TRANSFER FUNCTION  (OLTF)

                    OLK   =   GAIN*(OLNK/OLDK)   =   1.000

                           GAIN=     1.000

                           OLTF(S)   NUMERATOR
I      OLNPOLY(I)                                    OLZERO(I)
1   ( 1.000    )S** 1           ( -3.000   ) + J(  0.0000E+00)
2   ( 3.000    )                                OLNK=   1.000

                          OLTF(S)   DENOMINATOR
I      OLDPOLY(I)                                    OLPOLE(I)
1   (  1.000     )S** 2        ( -2.000   ) + J(  0.0000E+00)
2   (  3.000     )S** 1        ( -1.000   ) + J(  0.0000E+00)
3   (  2.000     )                                 OLDK=    1.000


ENTER YOUR A,B,C,D MATRICES : A,B2,C1,D
```

```
                    OPEN-LOOP TRANSFER FUNCTION  (OLTF)

                    OLK  = GAIN*(OLNK/OLDK)  =  4.000

                           GAIN=     1.000

                         OLTF(S)  NUMERATOR
I      OLNPOLY(I)                          OLZERO(I)
1  (   1.000     )S** 1          (  -1.000  ) + J(  0.0000E+00)
2  (   1.000     )                       OLNK=     4.000

                         OLTF(S)  DENOMINATOR
I      OLDPOLY(I)                          OLPOLE(I)
1  (   1.000     )S** 2          (  -2.000  ) + J(  0.0000E+00)
2  (   3.000     )S** 1          (  -1.000  ) + J(  0.0000E+00)
3  (   2.000     )                       OLDK=     1.000

ENTER YOUR A,B,C,D MATRICES : A,B2,C2,D

                    OPEN-LOOP TRANSFER FUNCTION  (OLTF)

                    OLK  = GAIN*(OLNK/OLDK)  =  1.000

                          GAIN=    1.000

                         OLTF(S) NUMERATOR
I      OLNPOLY(I)                          OLZERO(I)
1  (   1.000     )S** 1        (  -1.000  ) + J(  0.0000E+00)
2  (   1.000     )                     OLNK=      1.000

                        OLTF(S) DENOMINATOR
I      OLDPOLY(I)                          OLPOLE(I)
1  (   1.000     )S** 2        (  -2.000  ) + J(  0.0000E+00)
2  (   3.000     )S** 1        (  -1.000  ) + J(  0.0000E+00)
3  (   2.000     )                     OLDK=      1.000

ENTER YOUR A,B,C,D MATRICES : $

ICECAP> STOP
```

Example 2. This example illustrates the use of Riccati solver function to solve optimal control problem using the assigned problem 15-3 on page 714 [34].

Problem : Find the feedback coefficient matrix for the system and PI indicated:

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -6 & -11 & -6 \end{bmatrix} ; \quad B = \begin{bmatrix} 0 \\ 0 \\ 10 \end{bmatrix} ; \quad Y = \begin{bmatrix} 1 & 2 & 0 \end{bmatrix} X$$

$$PI = \int_0^\infty (X_1^2 + 0.01X_2^2 + 0.01X_3^2 + zU^2)\, dt$$

(a) $z = 1$, (b) $z = 10$, (c) $z = 0.1$. (d) Compare the time responses of the system with a step input for each value of z

```
ICECAP> DEFINE MATRIX
[>: A = <0 1 0;0 0 1;-6 -11 -6>,B=<0 0 10>;B = B',C=<1 2 0>
A     =
          0.    1.    0.
          0.    0.    1.
         -6.  -11    -6.

B     =
          0.
          0.
         10.

C     =
          1    2    0
[>: Q = <1 0 0;0 0.01 0;0 0 .01>,Z1 = 1,Z2 = 10,Z3 = 0.1
Q     =

          1.   0.      0.
          0.   0.01    0.
          0.   0.      0.01

Z1     =
          1.

Z2     =
          10.

Z3     =
          0.1

[>: $
```

```
ICECAP> DISPLAY RICCATI

ENTER YOUR A,B,Z,Q MATRICES IN THIS ORDER : A,B,Z3,Q

ANS    =

   -0.8613           0.2398          -0.0382
    0.2398          -0.2395           0.0137
   -0.0382           0.0137          -0.0146

The solution is not positive definite.
For positive definite solution, type YES for more answer.

SAVE THIS SOLUTION ? N

MORE ANSWERS ? Y

ANS    =

   -0.8613           0.2398          -0.0382
    0.2398          -0.2395           0.0137
   -0.0382           0.0137          -0.0146

The solution is not positive definite.
For positive definite solution, type YES for more answer.

SAVE THIS SOLUTION ? N

MORE ANSWERS ? Y

ANS    =

   0.7293            0.2398           0.0262
   0.2398            0.1195           0.0137
   0.0262            0.0137           0.0026

The solution is positive definite.

To save, name it : P3

Here is your feedback matrix

ANS    =

    2.6187           1.3710           0.2569

To save, name it : K

Here is your closed-loop matrix
```

```
ANS   =

      0.          1.          0.
      0.          0.          1.
    -32.187     -24.710      -8.569

To save, name it : ACL

MORE ANSWERS ? N

ENTER YOUR A,B,Z,Q MATRICES : $

ICECAP> DISPLAY MATRIX

[>: LONG R,K = K'

 K    =

     2.618695387886217
     1.371016137664886
     0.256856596831102

[>: PRINT('EX2',P3),PRINT('EX2',K),PRINT('EX2',ACL)

[>: $

ICECAP> DISPLAY TRANSFER/F

ENTER YOUR A,B,C,D MATRICES : ACL,B,C,D

                   OPEN-LOOP TRANSFER FUNCTION (OLTF)

                   OLK   =  GAIN*(OLNK/OLDK)=  20.00

                              GAIN=     1.000

                             OLTF(S) NUMERATOR
I       OLNPOLY(I)                                 OLZERO(I)
1 .  (   1.000      )S** 1          (   -0.5000   ) + J(   0.0000E+00)
2    (   0.500      )                                  OLNK=     20.00

                             OLTF(S) DENOMINATOR
I       OLDPOLY(I)                                 OLPOLE(I)
1    (   1.000      )S** 3          (   -4.834    ) + J(   0.0000E+00)
2    (   8.569      )S** 2          (   -1.867    ) + J(  -1.781      )
3    (   24.71      )S** 1          (   -1.867    ) + J(   1.781      )
4    (   32.19      )                                 OLDK=     1.000
```

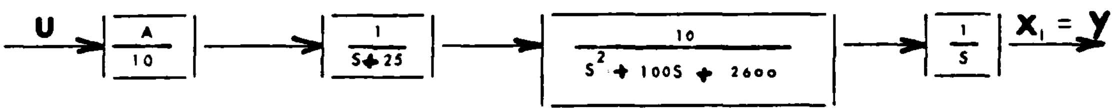
```
ENTER YOUR A,B,C,D MATRICES : $

ICECAP> COPY OLTF CLTF

ICECAP> DISPLAY SPECS

CONTINUOUS TIME RESPONSE FOR CLTF(S)
WITH STEP INPUT OF STRENGTH =    1.00000

     RISE TIME:           TR=      0.165856
     DUPLICATION TIME:    TD=      0.244345
     PEAK TIME:           TP=      0.759128
     SETTLING TIME:       TS=      3.24087
     PEAK VALUE:          MP=      0.766807
     FINAL VALUE:         FV=      0.310685

ICECAP> DISPLAY EQUATION

CONTINUOUS TIME RESPONSE FOR CLTF(S)
WITH STEP INPUT OF STRENGTH =    1.000000

THE TIME FUNCTION IS
F(T)=
        1.4971  EXP(-4.8345    T)
        2.8234  EXP(-1.8670    T) SIN( 1.7810 *T+  -39.814)
        0.31068  EXP(0.00000E+00T)

ICECAP> PRINT CLTF

ICECAP> PRINT SPECS

ICECAP> PRINT EQUATION

ICECAP> STOP
```

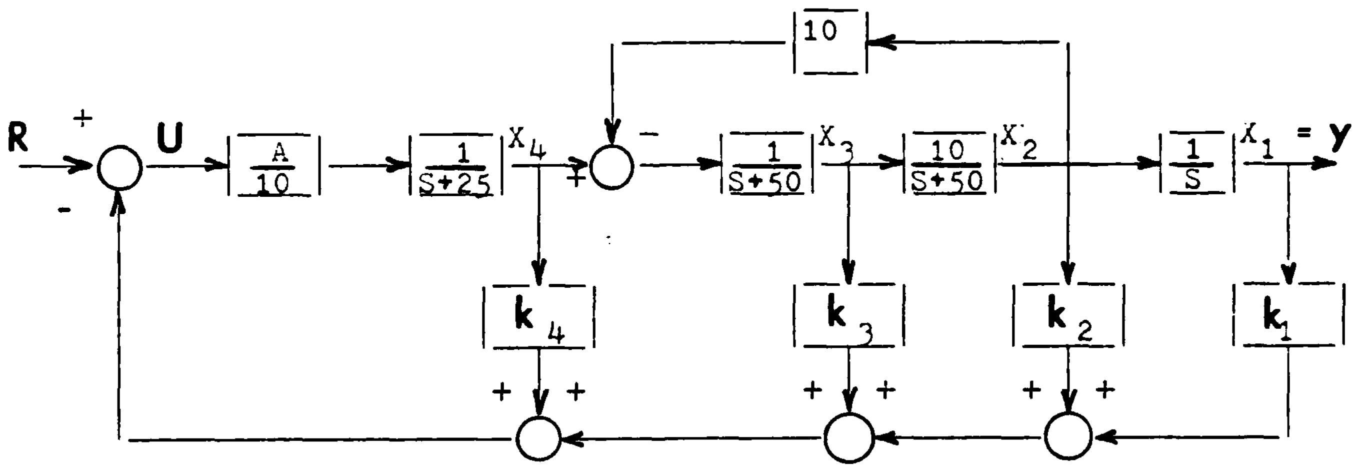
Note : To print the data out, try PRINT ANSWER.DAT,EX2.DAT

Example 3. This example illustrates the use of state variable feedback design function using the assigned problem 12-13 on page 708 [34].

Problem : Design a state-variable feedback system for the given plant $G_x(S)$. The desired complex dominant roots are to have a $\zeta = 0.425$. For a unit step function the

approximate specifications are $M_p = 1.2$, $t_p = 0.15$ s, and $t_s = 0.3$ s (a) Determine a desirable M(s) that will satisfy these specifications. Use steps 1 to 6 of the design procedure given in Sec. 12-5 to determine k. Draw the root locus for $G_{eq}(S)H(S) = -1$. From it show the "good" properties of state feedback. (b) Obtain y(t) for the final design. Determine the values of the figures of merit and the ramp-error coefficient.





$$\dot{X}_1 = X_2 \quad ; \quad \dot{X}_2 = -50X_2 + 10X_3$$

$$\dot{X}_3 = -50X_3 - 10X_2 + X_4 \quad ; \quad \dot{X}_4 = -25X_4 + U$$

$$\dot{X} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -50 & 10 & 0 \\ 0 & -10 & -50 & 1 \\ 0 & 0 & 0 & -25 \end{bmatrix} X + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} U$$

$$Y = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix} X$$

From desired specs, the model is

$$\frac{Y(S)}{R(S)}_D = \frac{7835831}{S + 206.6S + 13767.5S + 389106.2S + 7835831}$$

Use ICECAP to solve for k's

```
ICECAP> DEFINE CLTF POLY

POLYNOMIAL INPUT OF CLTF

ENTER NUM & DENOM DEGREES (OR SOURCE):   0,4
ENTER 1 NUMER COEFF--HI TO LO:   7835831

CLTF NUMERATOR (CLNPOLY)              CLTF ZEROS (CLZERO)
(  0.7836E+07)          POLYNOMIAL  CONSTANT=      0.7836E+07

ENTER 5 DENOM COEFF--HI TO LO: 1,206.6,13767.5,389106.2,7835831
```

```
CLTF DENOMINATOR (CLDPOLY)          CLTF POLES (CLPOLE)
(  1.000      )S** 4        (    -13.30       ) + J(      28.33    )
(  206.6      )S** 3        (    -13.30       ) + J(     -28.33    )
( 0.1377E+05)S** 2          (    -80.00       ) + J(   0.0000E+00)
( 0.3891E+06)S** 1          (    -100.0       ) + J(   0.0000E+00)
( 0.7836E+07)               POLYNOMIAL CONSTANT=        1.0000

  CLK= (CLNK/CLDK)=         7835831.

ICECAP> DEFINE MATRIX

[>: A=<0 1 0 0;0 -50 10 0;0 -10 -50 1;0 0 0 -25>,B=<0 0 0 1>;B=B'

A     =

   0.     1.     0.     0.
   0.   -50.    10.     0.
   0.   -10.   -50.     1.
   0.     0.     0.   -25.


B     =

   0.
   0.
   0.
   1.

[>: C= 1 0 0 0

C     =

   1.    0.    0.    0.

[>: $

ICECAP> DISPLAY MODERN

ENTER YOUR A,B,C MATRICES : A,B,C
```

```
                    THESE ARE VALID MODELS
                ---> CLTF          ---> OLTF
                ---> GTF           ---> HTF
     ---> PICK ONE : CLTF

KFB   =

    1.0000     0.0110     0.0006     0.0001

SAVE FEEDBACK MATRIX ? Y

NAME IT : K

                    THESE ARE VALID MODELS
               ---> CLTF           ---> OLTF
               ---> GTF            ---> HTF
    ---> PICK ONE : $

ICECAP> DISPLAY SPECS

CONTINUOUS TIME RESPONSE FOR CLTF(S)
WITH STEP INPUT OF STRENGTH =    1.000000

     RISE TIME:            TR=    0.558339E-01
     DUPLICATION TIME:     TD=    0.963022E-01
     PEAK TIME:            TP=    0.136230
     SETTLING TIME:        TS=    0.290197
     PEAK VALUE:           MP=     1.19772
     FINAL VALUE:          FV=     1.00000

ICECAP> DISPLAY EQUATION

CONTINUOUS TIME RESPONSE FOR CLTF(S)
WITH STEP INPUT OF STRENGTH =     1.0000000

THE TIME FUNCTION IS
F(T)=
      1.3371    EXP(-13.300    T) SIN( 28.330    *T+ 203.744)
     -0.93265   EXP(-80.001    T)
      0.47101   EXP(-99.999    T)
      1.0000    EXP(0.00000E+00T)
```

```
ICECAP> DISPLAY MATRIX

[>: DIR

Your current variables are...

K     TMAT  C     B     A     EPS     FLOP     EYE
RAND

using     49 out of   10005 elements.

[>: K=K';LONG R,K

K     =

   1.000000000000000
   0.011048120122606
   0.000647665052562
   0.000104137016359

[>: PRINT('EX3',K)

[>: $

ICECAP> STOP

Note : To print the feedback matrix K, try PRINT EX3.DAT
```

VITA

Lieutenant Chiewcharn Narathong was born on 10 May 1958 in the city of Pranburi, Thailand. He graduated from the Armed Force Military Preparatory School in Bangkok in 1977. Upon graduation, he attended the Royal Thai Air Force Academy for one year. With an outstanding academic record, the Thai government then selected him to attend the Virginia Military Institute (VMI) at Lexington, Virginia in 1979. At VMI Lieutenant Narathong studied Electrical Engineering and mathematics. In May 1983 he graduated and received a Bachelor of Science in Electrical Engineering with a distinguished record and received top honors with a monitary award. Lieutenant Narathong is a member of Phi Kappa Phi and Sigma Pi Sigma and is listed in Who's Who Among Student in American Universities and Colleges.

Permanent Address : 5600 16th street N.W.
Washington, D.C. 20011

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

AD-A153249

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION: UNCLASSIFIED

1b. RESTRICTIVE MARKINGS:

2a. SECURITY CLASSIFICATION AUTHORITY:

2b. DECLASSIFICATION/DOWNGRADING SCHEDULE:

3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; distribution unlimited

4. PERFORMING ORGANIZATION REPORT NUMBER(S): AFIT/GE/ENG/84D-49

5. MONITORING ORGANIZATION REPORT NUMBER(S):

6a. NAME OF PERFORMING ORGANIZATION: School of Engineering

6b. OFFICE SYMBOL (If applicable): AFIT/ENG

7a. NAME OF MONITORING ORGANIZATION:

6c. ADDRESS (City, State and ZIP Code): Air Force Institute of Technology, Wright-Patterson AFB, Ohio 45433

7b. ADDRESS (City, State and ZIP Code):

8a. NAME OF FUNDING/SPONSORING ORGANIZATION:

8b. OFFICE SYMBOL (If applicable):

9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER:

8c. ADDRESS (City, State and ZIP Code):

10. SOURCE OF FUNDING NOS.

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT NO.

11. TITLE (Include Security Classification): See Box 19

12. PERSONAL AUTHOR(S): Narathong Chiewcharn B.S., 2d Lt. Royal Thai Air Force

13a. TYPE OF REPORT: MS THESIS

13b. TIME COVERED: FROM ______ TO ______

14. DATE OF REPORT (Yr., Mo., Day): 1984 December

15. PAGE COUNT: 145

16. SUPPLEMENTARY NOTATION:

17. COSATI CODES

FIELD	GROUP	SUB. GR.
09	02	

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number): Riccati Solver, Eigen Number Approach, MATLAB, Modern Control, State Feedback Design, C(SI - A) B Function

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Title: A Modern Control Theory Enhancement To An Interactive Control Engineering Computer Analysis Package (ICECAP)

Thesis Chairman: Dr. Robert E. Fontana, Professor, AFIT
Dr. John Jones, Jr., Professor, AFIT

Approved for public release: IAW AFR 190-17.
LYNN E. WOLAVER [illegible]
Dean for Research and Professional Development
Air Force Institute of Technology (ATC)
Wright-Patterson AFB OH 45433

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT: UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS ☐

21. ABSTRACT SECURITY CLASSIFICATION: UNCLASSIFIED

22a. NAME OF RESPONSIBLE INDIVIDUAL: Dr. Robert E. Fontana, Professor, AFIT

22b. TELEPHONE NUMBER (Include Area Code): [illegible]

22c. OFFICE SYMBOL: AFIT/ENG

DD FORM 1473, 83 APR EDITION OF 1 JAN 73 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

This thesis reports on a continued development of an Interactive Control Engineering Computer Analysis Package (ICECAP). This package applies to discrete and continuous systems. The effort developed and implemented functional requirements involved in modern control theory. This includes implementing matrix operations, a continuous time steady state Riccati solver, polynomial operations, a $C(SI-A)^{-1}B$ function, and a state variable feedback design function.

Major emphasis is placed on the development of the Riccati solver. The approach employed is the eigen number approach. It provides all solutions which is not the case for other methods contained in the current literature. Additionally, A user friendly interface is also developed which permits interactive communication, user assistance and error analysis. Furthermore, the use of keyword/command word interactive technique is also employed. Lastly, and more importantly, the implementation of a state variable feedback design function allows the user to easily apply a pole placement state variable feedback design technique.

END

FILMED

5-85

DTIC